

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA E DE ARTES



CHATGPT: UMA ANÁLISE DA FERRAMENTA APLICADA NO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

Victor Junio Lisboa Costa

Goiânia

2024

CURSO SUPERIOR DE CIÊNCIA DA COMPUTAÇÃO

Victor Junio Lisboa Costa

**CHATGPT: UMA ANÁLISE DA FERRAMENTA APLICADA
NO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE**

Trabalho apresentado para o Curso de Ciência da Computação, da Pontifícia Universidade Católica de Goiás como parte dos requisitos para avaliação da unidade curricular de TCC.

Orientador(a): Prof. Msc Lucilia Ribeiro

Goiânia

2024

Victor Junio Lisboa Costa

CHATGPT: UMA ANÁLISE DA FERRAMENTA APLICADA NO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

Trabalho apresentado para o Curso de Ciência da Computação, da Pontifícia Universidade Católica de Goiás como parte dos requisitos para avaliação da unidade curricular de TCC.

Goiânia, 19 de Junho de 2024:

Prof. Msc Lucilia Ribeiro
Orientador

Prof. Msc Aníbal Santos Jukemura
Avaliador

Prof. Msc Fernando Gonçalves Abadia
Avaliador

Goiânia
2024

Agradecimentos

Gostaria de expressar minha profunda gratidão a todos os professores que fizeram parte da minha trajetória acadêmica na PUC Goiás. Em especial, agradeço à Professora Lucília Ribeiro, cuja orientação e apoio foram fundamentais para a conclusão do meu projeto de TCC.

A jornada acadêmica é repleta de desafios, e contar com professores dedicados e inspiradores faz toda a diferença. Cada aula, cada conselho e cada orientação contribuíram para o meu crescimento pessoal e profissional. A Professora Lucília Ribeiro, em particular, teve um papel crucial, oferecendo não apenas seu vasto conhecimento, mas também incentivo constante e orientação cuidadosa.

A todos os professores da PUC Goiás, meu sincero agradecimento por terem compartilhado seus conhecimentos e experiências, ajudando a moldar minha formação acadêmica e profissional. Cada um de vocês deixou uma marca indelével em minha trajetória, e sou eternamente grato por isso.

*Este trabalho, além de cultural, filosófico e pedagógico
É também medicinal, preventivo e curativo
Servindo entre outras coisas para pano branco e pano preto
Curuba e ferida braba
Piolho, chulé e caspa
Cravo, espinha e berruga
Panarismo e água na pleura
Só não cura o velho chifre
Por que não mata a raiz
Pois fica ela encravada
No fundo do coração
(Falcão)*

Resumo

Este Trabalho de Conclusão de Curso tem como objetivo explorar e aprimorar o processo de desenvolvimento de software no contexto empresarial, com foco na implementação de ferramentas e metodologias modernas. O estudo aborda as vantagens do uso de frameworks como o Spring Boot para a criação de aplicativos Java robustos e escaláveis, além de destacar a importância de ferramentas de teste e documentação de APIs, como o Postman, e *frameworks* de testes automatizados, como o *JUnit*. A metodologia adotada inclui uma revisão bibliográfica e a aplicação prática de tais ferramentas em projetos reais. Os resultados demonstram uma melhoria significativa na eficiência do desenvolvimento e na qualidade do software produzido. Conclui-se que a adoção de tecnologias adequadas e práticas recomendadas pode transformar positivamente o ambiente de desenvolvimento de software nas empresas.

Palavras-chave: Desenvolvimento de Software. IA. ChatGPT.

Abstract

This Final Course Project aims to explore and enhance the software development process in the business context, focusing on the implementation of modern tools and methodologies. The study discusses the advantages of using frameworks such as Spring Boot for creating robust and scalable Java applications, and highlights the importance of API testing and documentation tools like Postman, as well as automated testing frameworks like JUnit. The adopted methodology includes an in-depth literature review and the practical application of these tools in real projects. The results show a significant improvement in development efficiency and the quality of the produced software. It is concluded that adopting appropriate technologies and best practices can positively transform the software development environment in companies.

Keywords: Software Development. AI. ChatGPT.

Lista de ilustrações

Figura 1 – Rede neural	19
Figura 2 – Neurônio e rede neural	20
Figura 3 – Classificação PLN	22
Figura 4 – User story 1	53
Figura 5 – User story 2	53
Figura 6 – User story 3	54
Figura 7 – User story 4	54
Figura 8 – User story 5	55
Figura 9 – User story 6	55
Figura 10 – User story 7	56
Figura 11 – User story 8	56
Figura 12 – Estrutura do projeto	57
Figura 13 – Diagrama de caso de uso	58
Figura 14 – Diagrama de classes PedidoCafe e PreparandoCafe	59
Figura 15 – Diagrama de classes Pedidos e Pagamentos	59
Figura 16 – Diagrama de classes ItemPedidos	60
Figura 17 – User story - requisito 1	74
Figura 18 – Prompt para criar classe Pedido utilizando ChatGPT	76
Figura 19 – Prompt para criar método para atualizar pedido	78
Figura 20 – Prompt para criar teste de caso de uso	80
Figura 21 – Requisição HTTP para criar um novo pedido	82
Figura 22 – Resposta retornada HTTP para criar um novo pedido	83
Figura 23 – <i>Build</i> do projeto	83

Lista de abreviaturas e siglas

ACID	Atomicidade, Consistência, Isolamento, Durabilidade
AI	Artificial Intelligence
ANN	Redes Neurais Artificiais
API	Application Interface
BERT	Bidirectional Encoder Representations from Transformers
CI-CD	Continuous Integration - Continuous Development
CNN	Redes Neurais Convolucionais
CRUD	Create, Read, Update, Delete
DevOps	Development and Operation
DSDM	Dynamic System Development Method
DSRPAI	Dartmouth Summer Research Project on Artificial Intelligence
FDD	Feature Driven Development
GLN	Geração de Linguagem Natural
GPT	Generative Pre-trained Transformer
GPU	Unidades de Processamento Gráfico
HTTP	Hypertext Transfer Protocol
IA	Inteligência Artificial
IDE	Integrated Development Environment
IoT	Internet of Things
JSON	JavaScript Object Notation
JUnit	Open Source Java Unit Testing Tool
JPMS	Java Platform Module System
JVM	Java Virtual Machine
LSD	Lean Software Development

LSTMs	Long Short-Term Memory
ML	Machine Learning
PLN	Processamento de Linguagem Natural
RAD	Rapid Application Development
REST	Representational State Transfer
RLHF	Reinforcement Learning with Human Feedback
RNNs	Redes Neurais Recorrentes
RoBERTa	Robustly optimized BERT approach
RUP	Rational Unified Process
SGBDR	Sistema de Gerenciamento de Banco de Dados Relacional
T5	Text-To-Text Transfer Transformer
UML	Unified Modeling Language
URI	Uniform Resource Identifier
UUID	Identificador Único Universai
XML	Extensible Markup Language

Sumário

1	Introdução	12
2	Fundamentação Teórica	13
2.1	INTELIGÊNCIA ARTIFICIAL	13
2.1.1	HISTÓRIA E EVOLUÇÃO	16
2.1.2	REDES NEURAIS	18
2.1.3	PROCESSAMENTO DE LINGUAGEM NATURAL	21
2.1.4	<i>TRANSFORMERS</i> E MODELOS DE LINGUAGEM	24
2.1.5	<i>DEEP LEARNING</i>	26
2.1.6	CHATGPT: ENTENDENDO A TECNOLOGIA	26
2.2	DESENVOLVIMENTO DE SOFTWARE	29
2.2.1	PROCESSO DE DESENVOLVIMENTO DE SOFTWARE	34
2.2.2	FERRAMENTAS DE DESENVOLVIMENTO DE SOFTWARE	40
2.2.2.1	JIRA	41
2.2.2.2	JAVA	42
2.2.2.3	JUNIT	43
2.2.2.4	POSTMAN	44
2.2.2.5	SPRING BOOT	45
2.2.2.6	H2	46
2.2.3	TIPOS DE DESENVOLVIMENTO DE SOFTWARE	46
2.2.4	METODOLOGIAS ÁGEIS APLICADAS NO DESENVOLVIMENTO DE SOFTWARE	47
2.2.5	APLICAÇÃO DO DESENVOLVIMENTO DE SOFTWARE NO CONTEXTO DO CHATGPT	49
3	Metodologia	51
3.0.1	QUESTÕES DE PESQUISA	51
3.0.2	MODELAGEM DO SOFTWARE	51
3.0.2.1	REQUISITOS	52
3.0.2.2	ARQUITETURA DO SOFTWARE	57
3.0.2.3	CAMADA DE DOMÍNIO	58
3.0.2.4	CAMADA DE INFRAESTRUTURA	68
4	Apresentação e Análise de Resultados	74
4.0.1	CHATGPT NA COLETA DE REQUISITOS	74

4.0.2	CHATGPT NO PROCESSO DE DESENVOLVIMENTO DE CÓDIGO E TESTES	75
4.1	RESULTADOS	81
5	Considerações Finais	84
	Referências	86

1 Introdução

No cenário atual do desenvolvimento de software, a utilização de ferramentas de inteligência artificial tem ganhado destaque, especialmente pela sua capacidade de otimizar processos e auxiliar profissionais em diversas etapas do desenvolvimento. Este trabalho tem como objetivo analisar o impacto do uso do ChatGPT, no desenvolvimento de uma aplicação para uma cafeteria. Para isso, foram realizados experimentos que envolvem a geração de código, a modelagem de requisitos e a implementação de funcionalidades, destacando tanto os benefícios quanto as limitações encontradas.

No primeiro capítulo, apresentamos o contexto e a motivação do estudo, ressaltando a crescente demanda por soluções tecnológicas que agilizem o desenvolvimento de software e melhorem a eficiência dos profissionais da área. Exploramos como a inteligência artificial, particularmente o ChatGPT, pode ser integrada nesse processo, potencializando a produtividade e a qualidade dos projetos desenvolvidos.

No segundo capítulo, discutimos os aspectos teóricos que fundamentam o uso de IA no desenvolvimento de software. Abordamos conceitos relacionados à arquitetura de software, metodologias ágeis e boas práticas de programação, estabelecendo um embasamento sólido para a análise prática realizada posteriormente.

O terceiro capítulo é dedicado à metodologia adotada no projeto. Optamos por utilizar a arquitetura hexagonal, conhecida por sua modularidade e facilidade de teste, como estrutura para a aplicação desenvolvida. Descrevemos detalhadamente o processo de modelagem de requisitos utilizando o ChatGPT, bem como a implementação das funcionalidades principais da aplicação, como a gestão de pedidos e pagamentos .

No quarto capítulo, apresentamos uma análise crítica dos resultados obtidos. Avaliamos a eficiência do ChatGPT na geração de código e na modelagem de requisitos, destacando suas contribuições e identificando áreas onde a intervenção humana ainda é essencial. Observamos que, embora a ferramenta tenha facilitado diversas tarefas, a expertise do desenvolvedor continua sendo crucial para a interpretação e ajuste das sugestões geradas pela IA.

Finalmente, Este estudo oferece uma visão crítica sobre a integração da Inteligência Artificial no desenvolvimento de software, proporcionando insights para profissionais da área e contribuindo para a evolução das práticas de engenharia de software.

2 Fundamentação Teórica

2.1 INTELIGÊNCIA ARTIFICIAL

A Inteligência Artificial (IA) tem emergido como uma das áreas mais propícias e impactantes da ciência da computação. Em sua essência, a IA busca replicar processos cognitivos humanos, como aprendizado, raciocínio e tomada de decisão, utilizando algoritmos e modelos computacionais. Esse campo explora como os computadores podem aprender e realizar tarefas que anteriormente eram exclusivas e feitas manualmente por humanos, levando a avanços significativos em uma ampla gama de indústrias e aplicações. Desde sistemas de reconhecimento de padrões e veículos autônomos até assistentes virtuais e diagnósticos médicos, a IA está moldando o futuro da tecnologia e transformando a maneira como interagimos com o mundo ao nosso redor.

A evolução de IA está criando uma revolução tecnológica sem precedentes, redefinindo os limites do que é possível para a computação e para a sociedade como um todo. À medida que estes algoritmos de IA se tornam mais sofisticados e as máquinas que aprendem com grandes conjuntos de dados cada vez mais rápido por conta da evolução dos últimos anos dos computadores, somos confrontados com novas oportunidades e desafios. É crucial explorar não apenas os benefícios potenciais da IA, como a automação de tarefas tediosas e a melhoria da eficiência, mas também considerar cuidadosamente suas implicações éticas, sociais e econômicas. Compreender o impacto da IA e orientar seu desenvolvimento de forma responsável é essencial para garantir que ela beneficie a humanidade de maneira significativa e sustentável.

Para [Capozzi \(2023\)](#) é de extrema importância salientar que IA já integra significativamente nossas vidas há bastante tempo, frequentemente atuando de forma imperceptível para nós como um benefício oculto. Ao conduzir pesquisas na internet, receber recomendações personalizadas em serviços de streaming ou sermos direcionados para conteúdo específico em redes sociais, estamos, de fato, imersos na influência da IA.

Certamente, a inteligência artificial se estabeleceu como uma presença significativa em nossas vidas, mesmo que frequentemente não estejamos conscientes de seu impacto direto. Desde a personalização de recomendações de produtos em lojas online até a otimização de rotas de transporte em aplicativos de navegação, a IA está constantemente trabalhando nos bastidores para melhorar nossa experiência cotidiana.

Nesse contexto, a comparação feita por [Kai-Fuu \(2019\)](#) que equipara a IA ao aproveitamento da eletricidade de Thomas Edison, é extremamente pertinente. Assim como a eletricidade revolucionou múltiplas indústrias ao longo do tempo, a IA representa uma tecnologia inovadora com potencial para transformar não apenas uma, mas dezenas de

indústrias diferentes (apud NG ANDREW, 2017).

[Aggarwal et al. \(2022\)](#) reforçam a ideia, vivemos em época em que a IA está crescendo significativamente. Em breve, a IA se tornara um companheiro indispensável do homem, integrando-se cada vez mais em nossa vida e processos cotidianos.

À medida que a IA se solidifica como uma presença significativa em nossa sociedade, é evidente que seu impacto vai além das esferas industriais e do cotidiano humano. A analogia entre a IA e a eletricidade de Thomas Edison, proposta por [Kai-Fuu \(2019\)](#), ressalta a magnitude desse fenômeno. Assim como a eletricidade catalisou avanços em diversas áreas, desde a comunicação até a produção industrial, a IA está impulsionando uma nova era de descobertas e inovações.

Conforme [Aggarwal et al. \(2022\)](#) destacou anteriormente, estamos testemunhando um momento de crescimento exponencial da IA. A integração cada vez mais profunda da IA em áreas como pesquisa científica, medicina e educação promete não apenas transformar nossas capacidades e processos, mas também expandir os limites de nosso conhecimento e compreensão do mundo. À medida que nos aproximamos de um futuro onde a IA se tornará um companheiro indispensável, é crucial não apenas abraçar suas promessas, mas também enfrentar os desafios éticos e sociais que ela apresenta. Somente assim será possível aproveitar plenamente o potencial transformador da inteligência artificial.

Para se capacitar um computador a desenvolver habilidades semelhantes às humanas, como aprendizado, raciocínio e tomada de decisão, é fundamental que ele possua algumas capacidades básicas. [Russell e Norvig \(2013\)](#) diz que uma IA é composta de seis características básicas:

1. Processamento de Linguagem Natural (PLN) para permitir se comunicar em forma de linguagem natural e não de máquina.
2. Representação do Conhecimento para armazenar informações adquiridas por meio da percepção ou aprendizado.
3. Raciocínio automatizado para utilizar as informações armazenadas a fim de responder a perguntas e deduzir novas conclusões de forma autônoma.
4. Aprendizado de máquina (Machine Learning ML) para se ajustar a novos cenários e discernir padrões, adaptando-se dinamicamente as mudanças e extrapolando informações relevantes.
5. Visão computacional para identificar e compreender objetos, permitindo uma compreensão mais abrangente do ambiente ao redor.
6. Robótica para manipular objetos com destreza e se locomover com eficiência e precisão.

Após estabelecermos as bases essenciais da inteligência artificial, é importante destacar que essas tecnologias interagem de maneira sinérgica, formando um ecossistema complexo e interconectado. Por exemplo, a visão computacional pode ser integrada à robótica para permitir que um robô reconheça objetos em seu ambiente e os manipule com precisão.

De acordo com [Kai-Fuu \(2019\)](#) ao fornecer as máquinas com habilidades como visão, tato e capacidade de otimização de dados, ampliamos consideravelmente o leque de tarefas que elas podem resolver, como por exemplo, uma máquina capaz de colher morangos poderia constantemente aprender com o ambiente, aprimorando sua eficiência e precisão ao longo do tempo.

Da mesma forma, o PLN pode ser combinado com o aprendizado de máquina para criar sistemas de conversação avançados capazes de responder a perguntas de forma inteligente e natural. Seguindo essa ideia, [Liu, Lin e Sun \(2023\)](#) afirmam que o PLN pode ser usado na tradução simultânea de idiomas, visando, assim, superar as barreiras linguísticas entre pessoas de todo o mundo. Essa abordagem mostra como a tecnologia de PLN pode ter um impacto significativo na comunicação global, permitindo que pessoas de diferentes partes do mundo se entendam e se conectem mais facilmente. A tradução simultânea de idiomas é apenas uma das muitas aplicações possíveis do PLN, destacando seu potencial para promover a compreensão intercultural e facilitar a colaboração em um mundo cada vez mais interconectado.

Além disso, o aprendizado de máquina pode ser aplicado em conjunto com a representação do conhecimento para aprimorar a capacidade de uma máquina de extrair *insights* significativos de grandes conjuntos de dados. Em uma pesquisa prática feita por [Dogan et al. \(2021\)](#), os autores demonstraram como o aprendizado de máquina poderia ter sido amplamente utilizada durante a pandemia de COVID-19 para detectar moléculas de COVID-19 em pacientes que tinham sintomas da doença. Muitos cientistas de dados recorrem a ferramentas de IA e ML para descobrir novos medicamentos, utilizando processamento de imagens em raios-X e tomografias computadorizadas (CT), a fim de identificar indivíduos infectados.

Neste capítulo mergulhamos no mundo das redes neurais e seu papel crucial no campo da IA. Este capítulo destaca como esses sistemas, inspirados pela complexidade do cérebro humano, transformaram o aprendizado de máquina. Abordamos desde as bases conceituais até as inovações modernas, incluindo tipos variados de redes e sua evolução histórica. Além disso, discutimos como as redes neurais são aplicadas em cenários reais, demonstrando seu impacto e relevância atual. Em seguida, focamos no aprendizado de máquina, uma pedra angular da IA, onde modelos computacionais são treinados para identificar padrões e tomar decisões. Aqui, desvendamos diferentes abordagens de aprendizado, como supervisionado e não-supervisionado, e como essas técnicas moldam aplicações práticas. Particular atenção é dada ao processamento de linguagem natural, destacando como essa tecnologia revolucionou

a compreensão e geração de linguagem em IA, exemplificado pelo desenvolvimento do ChatGPT.

2.1.1 HISTÓRIA E EVOLUÇÃO

A história e evolução da inteligência artificial remontam a décadas atrás, com raízes profundas em conceitos teóricos e avanços tecnológicos. Desde os primórdios da computação, os seres humanos têm buscado replicar o pensamento e a inteligência em máquinas, levando a uma jornada de descobertas e inovações que moldaram o cenário atual da IA. A história da inteligência artificial é marcada por avanços significativos em áreas como aprendizado de máquina, processamento de linguagem natural, visão computacional e robótica, cada um contribuindo para ampliar os horizontes do que é possível para as máquinas. Ao explorarmos essa fascinante narrativa, é possível compreender não apenas como a IA se tornou uma parte essencial de nossas vidas, mas também antecipar as direções futuras desse campo em constante evolução.

A história da Inteligência Artificial segundo [Russell e Norvig \(2013\)](#), remonta ao trabalho pioneiro de Warren McCulloch e Walter Pitts (1943), agora reconhecido como o marco inicial desse campo. Sua pesquisa foi fundamentada em três pilares: o conhecimento da fisiologia básica e da função dos neurônios no cérebro humano; uma análise formal da lógica proposicional criada por Russell e Whitehead; e a teoria da computação de Turing. Os pesquisadores propuseram um modelo de neurônios artificiais, nos quais cada neurônio pode estar "ligado" ou "desligado", ativando-se quando estimulado por um número suficiente de neurônios vizinhos, demonstrando que esses neurônios podem realizar cálculos computacionais e implementar conectivos lógicos simples como (e, ou, não).

Após o trabalho seminal de Warren McCulloch e Walter Pitts (1943), a história da Inteligência Artificial ganhou novos contornos com a contribuição de outros importantes pesquisadores. Destaca-se o papel de Alan Turing, cuja teoria da computação desempenhou um papel crucial no desenvolvimento da IA.

De acordo com [Smith et al. \(2006\)](#) em 1950 o matemático inglês Alan Turing publicou um artigo intitulado *Computer Machinery and Intelligence* (Máquina de Computação e Inteligência), que abriu portas para o campo que hoje seria chamado de IA. Isso ocorreu anos antes de a comunidade adotar o termo Inteligência Artificial. Alan Turing, em seu artigo questionou se as máquinas podem pensar e propôs o famoso teste de Turing como um método para avaliar essa capacidade. O teste sugere que uma máquina que se comporta indiferente de um ser humano inteligente pode ser considerada capaz de pensar.

Esse artigo de Turing desencadeou uma revolução na forma como encaramos a capacidade das máquinas de pensar e agir de forma inteligente. Seu conceito de teste de Turing desafiou as fronteiras entre humanos e máquinas, levantando questões fundamentais sobre

a natureza da inteligência e da consciência. A proposta de Turing não apenas estabeleceu as bases para o campo emergente da IA, mas também inspirou gerações de pesquisadores a explorar os limites da computação e a buscar maneiras de criar sistemas cada vez mais inteligentes e autônomos. Este teste ainda continua atualmente sendo de grande relevância. Ao lançar luz sobre esses debates fundamentais, Turing moldou não apenas o futuro da tecnologia, mas também nossa compreensão do que significa ser humano.

Em seu estudo sobre a história da evolução da Inteligência Artificial, [Haenlein e Kaplan \(2019\)](#) destacam um evento crucial que deixou sua marca na trajetória da IA. Cerca de seis anos após esse marco, em 1956, Marvin Minsky e John McCarthy, ambos pesquisadores renomados, organizaram o Projeto de Pesquisa de Verão sobre Inteligência Artificial (*Dartmouth Summer Research Project on Artificial Intelligence DSRPAI*) no *Dartmouth College*, em New Hampshire. Financiado pela Fundação *Rockefeller*, esse workshop de aproximadamente oito semanas marcou o início da era moderna da IA e reuniu figuras proeminentes que viriam a ser consideradas os pioneiros do campo. Entre os participantes estavam Nathaniel Rochester, renomado cientista da computação responsável pelo projeto do IBM 701, o primeiro computador científico comercial, e Claude Shannon, matemático influente e pioneiro na teoria da informação.

Após a Conferência de Dartmouth, conforme relatado por [Haenlein e Kaplan \(2019\)](#) em seu artigo, houve um período de quase duas décadas que testemunhou sucessos significativos no campo da IA. Um exemplo precoce é o famoso programa de computador ELIZA, desenvolvido entre 1964 e 1966 por Joseph Weizenbaum no MIT. ELIZA era uma ferramenta de processamento de linguagem natural capaz de simular uma conversa com um ser humano e um dos primeiros programas a tentar passar no mencionado Teste de Turing. Outro marco nos primeiros dias da IA foi o programa *General Problem Solver*, criado pelo ganhador do Prêmio Nobel Herbert Simon e pelos cientistas da RAND Corporation Cliff Shaw e Allen Newell, capaz de resolver automaticamente certos tipos de problemas simples, como as Torres de Hanói. O sucesso desses projetos inspiradores resultou em financiamento substancial para pesquisas em IA, impulsionando o desenvolvimento de mais e mais projetos na área.

Após uma década de grandes avanços no campo da IA, o período conhecido como Inverno da IA na década de 1980 marcou uma desaceleração significativa na pesquisa e desenvolvimento no campo da IA, conforme revelado por [Gold \(2023\)](#). Isso se deu em parte devido ao fato de que muitos dos projetos de IA desenvolvidos durante o boom inicial não conseguiram cumprir suas promessas.

A comunidade de pesquisa em IA ficou cada vez mais desiludida com a falta de progresso no campo, o que resultou em cortes de financiamento e muitos pesquisadores de IA sendo forçados a abandonar seus projetos. O Inverno da IA na década de 1980 foi caracterizado por uma queda significativa no financiamento para pesquisa em IA e uma falta

geral de interesse no campo entre investidores e o público. Isso levou a uma diminuição no número de projetos de IA em desenvolvimento, com muitos projetos de pesquisa ainda ativos incapazes de fazer progressos significativos devido à falta de recursos.

Ainda em seu artigo [Gold \(2023\)](#), destaca avanços significativos foram feitos na década de 1990, onde o uso de métodos estatísticos revolucionou o desenvolvimento de sistemas avançados de Processamento de Linguagem Natural e Visão Computacional. Ao invés de depender de regras pré-definidas, os pesquisadores passaram a utilizar algoritmos de aprendizado de máquina para aprender padrões diretamente dos dados. Essa nova abordagem possibilitou a criação de modelos mais precisos e flexíveis para lidar com informações visuais e linguísticas.

[Haenlein e Kaplan \(2019\)](#) destacam o ressurgimento da IA, exemplificado pelo AlphaGo do Google em 2015. Ao vencer o campeão mundial de Go, um jogo muito mais complexo que o xadrez, o programa desafiou a crença de que os computadores não poderiam vencer humanos nesse jogo. Esse evento foi um marco significativo, demonstrando a eficácia da IA em lidar com desafios complexos.

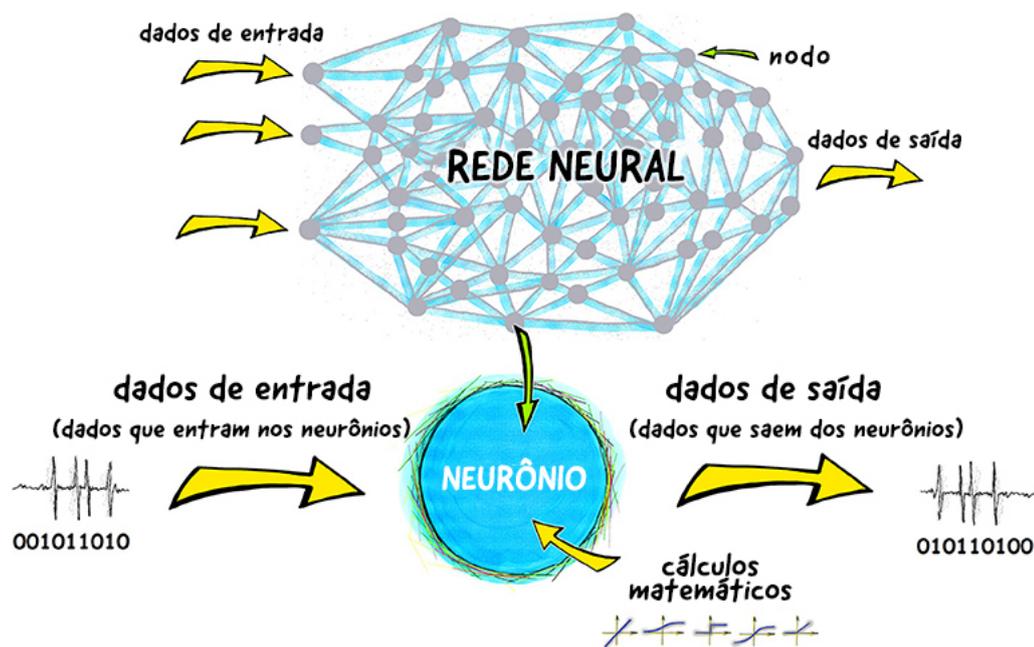
Apesar dos desafios enfrentados durante o Inverno da IA na década de 1980, a pesquisa em inteligência artificial deu uma guinada significativa nas décadas seguintes. Na virada do século, os avanços na computação e nos algoritmos abriram novas perspectivas para o desenvolvimento de sistemas mais sofisticados de Processamento de Linguagem Natural e Visão Computacional. O uso de métodos estatísticos e algoritmos de aprendizado de máquina revolucionou a abordagem, permitindo que os sistemas aprendessem diretamente dos dados e adaptassem-se de forma flexível a diferentes cenários. O ressurgimento da IA atingiu um marco significativo com o AlphaGo do Google em 2015, desafiando as noções convencionais sobre as capacidades dos computadores em lidar com desafios complexos. Esse evento destacou o potencial da IA em superar barreiras aparentemente intransponíveis, lançando luz sobre um futuro promissor para a Inteligência Artificial.

2.1.2 REDES NEURAIS

As redes neurais, inspiradas no funcionamento do cérebro humano, são um componente central da inteligência artificial moderna. Elas consistem em um conjunto de algoritmos e modelos matemáticos que imitam a estrutura e o funcionamento dos neurônios do cérebro. Essas redes são compostas por camadas de unidades interconectadas, chamadas de neurônios artificiais ou unidades de processamento, que trabalham em conjunto para realizar tarefas complexas, como reconhecimento de padrões, classificação de dados e tomada de decisões. O aprendizado ocorre através do ajuste dos pesos das conexões entre os neurônios, permitindo que a rede melhore seu desempenho à medida que é exposta a mais dados.

A figura 1 uma rede neural artificial é feita de neurônios artificiais, os nodos, que

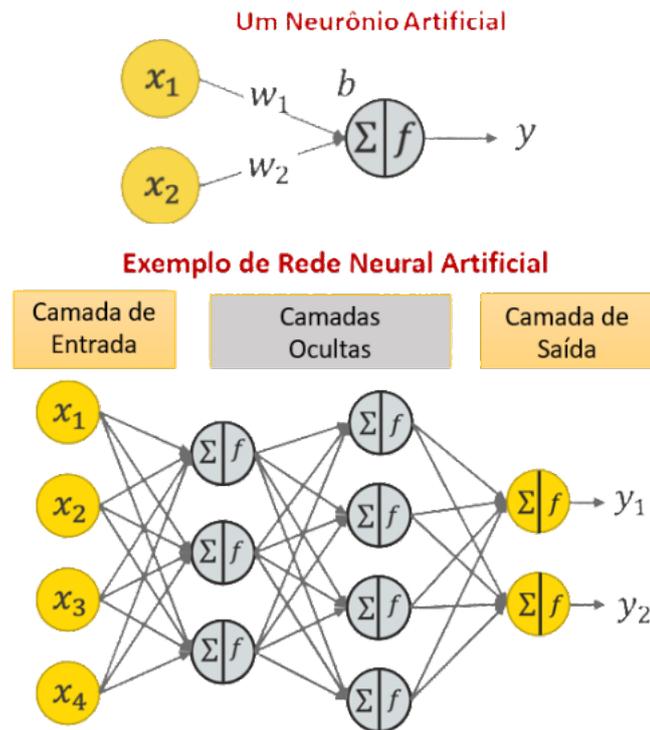
Figura 1 – Rede neural



Fonte: Unesp

fazem cálculos com dados de entrada e transmitem os resultados desses cálculos como dados de saída.

Figura 2 – Neurônio e rede neural



Fonte: [Nei Grando](#)

Um neurônio artificial é o menor bloco de construção em todas as redes neurais profundas. Em uma rede neural artificial, o neurônio artificial recebe como entrada um ou mais estímulos na forma de sinais que são números reais (x_1, x_2, \dots). A saída (y) de cada neurônio é calculada usando alguns pesos fixos (w_1, w_2, \dots), uma polarização (b) e uma função de ativação definida (f) não linear da soma de suas entradas Σ . ([GRANDO, 2022](#))

- As conexões entre os neurônios são chamadas de arestas.
- Tanto os neurônios quanto as arestas têm um peso. Este parâmetro se ajusta e muda à medida que o aprendizado prossegue.
- O peso aumenta ou diminui a força do sinal em uma conexão.
- Os neurônios podem ter um limiar. Um sinal é enviado somente se o sinal agregado cruzar esse limite.

Como explicado previamente, os neurônios formam camadas que processam entradas de maneira sequencial. Os sinais são transmitidos da camada de entrada para a camada de saída, com possíveis modificações ao longo do caminho, podendo atravessar as camadas várias vezes. As redes neurais possuem um mecanismo de aprendizado embutido, ajustando os pesos das conexões neurais com base nos padrões de entrada, de forma semelhante à

maneira como uma criança em desenvolvimento aprende a identificar animais através de exemplos visuais.

As redes neurais vêm em uma variedade de formas e arquiteturas, cada uma projetada para resolver diferentes tipos de problemas e lidar com diferentes tipos de dados. Exemplos de redes neurais:

- Redes Neurais Artificiais (ANNs): São as formas mais básicas, geralmente usadas para tarefas de classificação e regressão.
- Redes Neurais Convolucionais (CNNs): Especializadas em processar dados com uma grade topológica, como imagens.
- Redes Neurais Recorrentes (RNNs): Úteis para trabalhar com sequências de dados, como texto ou séries temporais.
- Além disso, existem também arquiteturas híbridas que combinam diferentes tipos de redes neurais para aproveitar o melhor de cada uma delas.

Em suma, as redes neurais são uma poderosa ferramenta inspirada no funcionamento do cérebro humano, capaz de aprender e reconhecer padrões complexos nos dados. Seu potencial de aplicação é vasto, abrangendo desde reconhecimento de voz e imagens até previsão de séries temporais e tomada de decisões. À medida que continuamos a aprimorar nossa compreensão dessas redes e desenvolvemos algoritmos mais avançados, pode-se esperar que elas desempenhem um papel cada vez mais central em várias áreas da tecnologia e da ciência.

A relação de redes neurais com este trabalho se dá através do uso do ChatGPT como uma ferramenta no processo de desenvolvimento de software. Utilizando essa tecnologia, Espera-se ser possível otimizar diversas etapas do desenvolvimento, como a geração de código, a resolução de problemas e a elaboração de documentação. A capacidade do ChatGPT de entender e responder a consultas complexas pode melhorar significativamente a eficiência e a qualidade de um trabalho, demonstrando o impacto positivo das redes neurais na prática de engenharia de software.

2.1.3 PROCESSAMENTO DE LINGUAGEM NATURAL

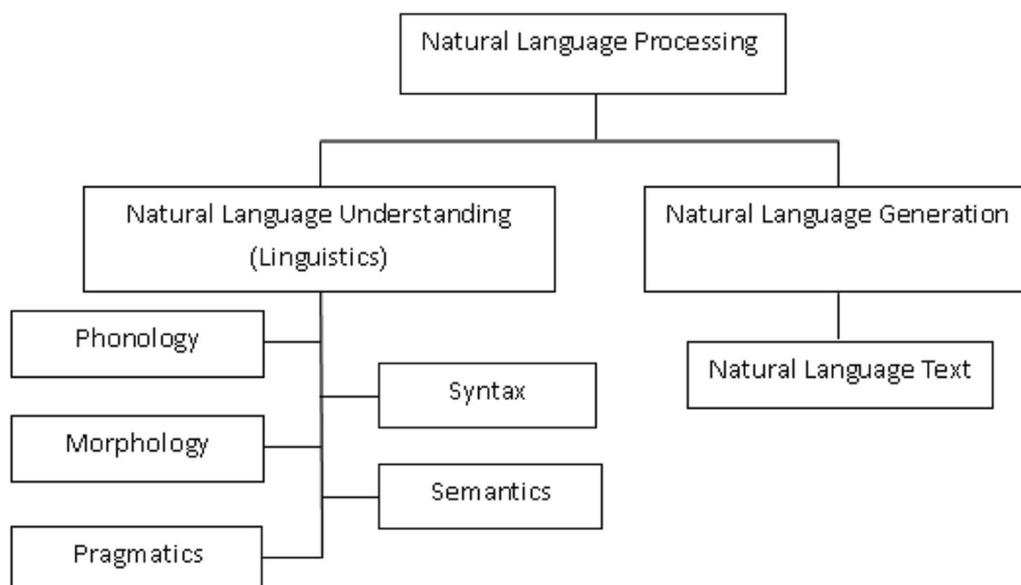
O Processamento de Linguagem Natural é uma área da inteligência artificial que se concentra na interação entre computadores e linguagem humana. Seu objetivo é capacitar os computadores a entender, interpretar e gerar texto ou fala de maneira semelhante aos seres humanos. O PLN abrange uma variedade de tarefas, incluindo análise de sentimentos, tradução automática, extração de informações, sumarização de texto e geração de linguagem

natural. Com o avanço da tecnologia e o aumento da disponibilidade de dados linguísticos, o PLN desempenha um papel cada vez mais importante em aplicativos do mundo real, como assistentes virtuais, sistemas de recomendação e análise de mídia social.

Apesar do crescente uso do processamento de linguagem natural, essa tecnologia enfrenta diversos desafios, incluindo lidar com a ambiguidade da linguagem natural, compreender contextos complexos e garantir a privacidade e segurança dos dados linguísticos. No entanto, com o contínuo desenvolvimento de algoritmos de aprendizado de máquina, técnicas avançadas de processamento de texto e grandes conjuntos de dados linguísticos disponíveis, o PLN está se tornando cada vez mais sofisticado e eficaz em diversas aplicações. Essa evolução promete revolucionar ainda mais a maneira como interagimos com a tecnologia e como ela nos auxilia em diversas tarefas do dia a dia.

Conforme mencionado por [Khurana et al. \(2022\)](#), o processamento de linguagem natural abrange uma variedade de tarefas, desde a análise de sentimentos até a geração de linguagem natural. Um aspecto fundamental do PLN é sua divisão em duas partes principais: Entendimento de Linguagem Natural e Geração de Linguagem Natural (GLN).

Figura 3 – Classificação PLN



Fonte: [Khurana et al. \(2022\)](#)

Na Figura 3, pode-se observar a classificação do processamento de linguagem natural e sua divisão em diversos tópicos fundamentais, cada um desempenhando um papel crucial na compreensão e na geração de linguagem.

- A fonologia estuda os padrões sistemáticos de sons na linguagem, incluindo sua organização e funcionamento. Ela investiga como os sons são produzidos, percebidos

e organizados em sistemas linguísticos para transmitir significado. Na análise fonológica, os sistemas de PLN examinam as unidades sonoras básicas, como fonemas e alofones, e suas relações dentro de uma língua específica. Isso é essencial para o reconhecimento preciso da fala, a síntese de voz e outras tarefas relacionadas à comunicação oral.

- A morfologia abrange o estudo da estrutura interna das palavras em uma língua. Ela lida com a análise das unidades mínimas de significado, chamadas morfemas, e como essas unidades se combinam para formar palavras. Além disso, a morfologia examina os processos de formação de palavras, como a derivação e a flexão, que afetam a forma das palavras e seu significado. Em resumo, a morfologia no PLN é fundamental para compreender como as palavras são formadas e como seu significado é expresso na linguagem natural.
- O componente lexical refere-se ao vocabulário de uma língua e ao conjunto de regras que governam o uso das palavras. Ele lida com a identificação e análise das palavras em um texto, incluindo a segmentação em tokens individuais, a lematização para reduzir palavras à sua forma base e a identificação de sinônimos e antônimos. Além disso, o componente lexical também inclui a atribuição de significado a palavras com base no contexto em que são usadas. Em suma, o componente lexical desempenha um papel crucial no entendimento e processamento da linguagem natural.
- O componente sintático refere-se à estrutura gramatical das sentenças em um texto. Ele lida com a análise da sintaxe das frases para identificar a relação entre as palavras e a organização das unidades linguísticas. Isso inclui a análise da ordem das palavras, a identificação de partes do discurso, como sujeito, verbo e objeto, e a análise da hierarquia das frases, como frases simples, compostas e complexas. O componente sintático é essencial para compreender a estrutura das sentenças e extrair significado delas.
- O componente semântico se concentra na compreensão do significado das palavras e das sentenças em um texto. Ele utiliza técnicas para interpretar o contexto das palavras e identificar relações semânticas entre elas, permitindo que os sistemas de PLN compreendam o significado subjacente das informações. Isso inclui a análise de sinonímia, antonímia, polissemia e outras relações semânticas para uma compreensão mais profunda do texto. O componente semântico é fundamental para uma interpretação precisa e eficaz da linguagem natural.
- O componente pragmático se concentra na interpretação do significado do texto com base no contexto situacional e nas intenções do usuário. Ele permite que os sistemas de PLN considerem fatores como ironia, sarcasmo, ambiguidade e inferência para uma compreensão mais precisa da linguagem natural. O entendimento pragmático é

essencial para uma interação eficaz entre humanos e máquinas em uma variedade de aplicações, como assistentes virtuais, chatbots e sistemas de reconhecimento de fala.

- A geração de linguagem natural é o processo pelo qual os sistemas computacionais produzem texto ou fala em linguagem humana. As características do GLN incluem a capacidade de transformar dados estruturados ou conceitos em expressões linguísticas coerentes e compreensíveis. Isso envolve a seleção e organização adequada das palavras, a geração de frases gramaticalmente corretas e a adaptação do estilo e tom de acordo com o contexto e a audiência.

A compreensão e análise da linguagem natural abrangem diversos componentes essenciais que trabalham em conjunto para tornar possível a interação entre humanos e máquinas. A morfologia, responsável pela estrutura das palavras, examina como elas são formadas e modificadas, enquanto o componente lexical lida com a identificação e interpretação do vocabulário em um texto, atribuindo significados com base no contexto. Por sua vez, o componente sintático analisa a estrutura gramatical das sentenças, identificando a relação entre as palavras e organizando as unidades linguísticas. Já o componente semântico foca no significado das palavras e frases, interpretando o contexto e as relações semânticas para uma compreensão mais profunda do texto. Por fim, o componente pragmático considera o contexto situacional e as intenções do usuário para uma interpretação precisa, enquanto a geração de linguagem natural permite que os sistemas produzam texto ou fala coerente e adaptável. Esses elementos se combinam para viabilizar uma interação eficaz entre humanos e máquinas em uma variedade de aplicações no campo do processamento de linguagem natural.

2.1.4 TRANSFORMERS E MODELOS DE LINGUAGEM

Transformers são uma classe de modelos de aprendizado de máquina que revolucionaram o campo do processamento de linguagem natural. Introduzido por [Vaswani et al. \(2017\)](#), o *Transformer* é um modelo que utiliza uma arquitetura baseada em mecanismos de atenção para processar sequências de dados, como texto, de maneira mais eficiente e precisa do que os modelos baseados em redes neurais recorrentes (*RNNs*) e convolucionais (*CNNs*) tradicionais. A arquitetura *Transformer* se destaca pela capacidade de processar *tokens* de entrada simultaneamente, ao invés de de forma sequencial, o que permite paralelismo e uma melhor escalabilidade.

O componente central do *Transformer* é o mecanismo de atenção, especificamente a atenção auto-regressiva (*self-attention*), que permite ao modelo focar em diferentes partes da sequência de entrada ao processar cada token. Este mecanismo permite que o modelo capture dependências de longo alcance e interações complexas dentro do texto. A arquitetura *Transformer* é composta por camadas de codificadores (*encoders*) e decodificadores

(*decoders*), embora modelos voltados exclusivamente para tarefas de compreensão de texto possam utilizar apenas a parte do codificador (DEVLIN et al., 2019).

Modelos de linguagem pré-treinados são modelos de PLN que foram treinados em grandes volumes de texto antes de serem ajustados para tarefas específicas. Este pré-treinamento é geralmente realizado em tarefas de modelagem de linguagem, onde o modelo aprende a prever a próxima palavra em uma sequência (pré-treinamento não supervisionado) ou a prever palavras mascaradas (pré-treinamento supervisionado). Este processo permite que o modelo adquira uma compreensão rica da linguagem, capturando nuances semânticas, sintáticas e contextuais.

O processo de pré-treinamento gera *embeddings* de palavras ou *tokens* que são altamente informativos e podem ser utilizados em uma variedade de tarefas de PLN, como tradução automática, análise de sentimento, resposta a perguntas e geração de texto. Após o pré-treinamento, os modelos são frequentemente ajustados (*fine-tuned*) em conjuntos de dados específicos para a tarefa desejada, melhorando seu desempenho de maneira significativa.

Nos últimos anos, vários modelos de linguagem baseados na arquitetura *Transformer* ganharam destaque devido ao seu desempenho excepcional em diversas tarefas de PLN. Alguns dos modelos mais populares incluem:

- *BERT (Bidirectional Encoder Representations from Transformers)*: Introduzido por Devlin et al. (2019), o *BERT* é um modelo de codificador bidirecional que captura o contexto de uma palavra considerando tanto o lado esquerdo quanto o direito da sequência de palavras. Isso permite uma compreensão mais profunda do contexto e significado das palavras.
- *GPT (Generative Pre-trained Transformer)*: Desenvolvido pela OpenAI, o *GPT* é um modelo de decodificador que se destaca na geração de texto. O *GPT-3*, a terceira versão deste modelo, contém 175 bilhões de parâmetros, tornando-se um dos maiores modelos de linguagem já criados (BROWN et al., 2020).
- *T5 (Text-To-Text Transfer Transformer)*: Proposto por Raffel et al. (2020), o *T5* trata todas as tarefas de PLN como problemas de conversão de texto em texto. Esta abordagem unificada facilita o treinamento e a adaptação do modelo para diversas tarefas.
- *RoBERTa (Robustly optimized BERT approach)*: Uma variação do *BERT* introduzida por Liu et al. (2019), que otimiza o treinamento do modelo *BERT* original utilizando mais dados e treinamento mais longo.

Estes modelos têm sido amplamente adotados na pesquisa acadêmica e na indústria, impulsionando avanços significativos no estado da arte do processamento de linguagem

natural.

2.1.5 DEEP LEARNING

Deep learning, ou aprendizagem profunda, é uma subárea do aprendizado de máquina que utiliza redes neurais artificiais com múltiplas camadas para modelar e entender dados complexos. Essas redes neurais são inspiradas na estrutura e no funcionamento do cérebro humano, sendo capazes de realizar tarefas complexas como reconhecimento de imagem, processamento de linguagem natural e diagnóstico médico.

Uma característica fundamental do deep learning é sua capacidade de aprendizado supervisionado e não supervisionado, o que permite que as redes neurais aprendam a partir de grandes volumes de dados. Segundo [Lecun, Bengio e Hinton \(2015\)](#), "o aprendizado profundo permite que modelos computacionais que são compostos de múltiplas camadas de processamento aprendam representações de dados com múltiplos níveis de abstração". Isso significa que as redes neurais podem extrair características relevantes dos dados brutos de forma automática, sem a necessidade de intervenção humana.

O treinamento de redes neurais profundas exige poder computacional significativo e grandes conjuntos de dados anotados. Conforme [Goodfellow, Bengio e Courville \(2016\)](#), "os avanços em hardware, como o uso de GPUs (unidades de processamento gráfico), e a disponibilidade de grandes bases de dados têm sido fatores críticos para o progresso do deep learning". Esses avanços tecnológicos permitiram que redes neurais complexas fossem treinadas de maneira eficiente, resultando em desempenho superior em diversas aplicações.

Além disso, o deep learning tem mostrado resultados promissores em áreas como visão computacional e reconhecimento de fala. [Krizhevsky, Sutskever e Hinton \(2012\)](#) demonstraram que uma rede neural convolucional profunda pode superar significativamente métodos tradicionais em tarefas de classificação de imagens. Esses resultados reforçam a importância do deep learning como uma ferramenta poderosa para a inteligência artificial.

Portanto, o deep learning representa um avanço significativo no campo do aprendizado de máquina, permitindo que máquinas aprendam e realizem tarefas complexas com precisão crescente. A combinação de grandes conjuntos de dados, poder computacional avançado e algoritmos sofisticados continua a impulsionar o desenvolvimento e a aplicação de redes neurais profundas em diversos domínios.

2.1.6 CHATGPT: ENTENDENDO A TECNOLOGIA

O ChatGPT é uma aplicação de IA desenvolvida pela *OpenAI* que utiliza modelos de linguagem natural para interagir com os usuários de maneira conversacional. Baseado na arquitetura *GPT*, o ChatGPT é capaz de compreender e gerar texto em linguagem natural,

permitindo interações que se assemelham a diálogos humanos. Este modelo tem sido amplamente utilizado em diversas aplicações, desde assistentes virtuais até ferramentas educacionais, oferecendo respostas coerentes e contextualmente relevantes a uma ampla gama de perguntas e solicitações.

A arquitetura do ChatGPT é baseada no *Transformer*, uma inovação significativa no campo do processamento de linguagem natural. O *Transformer*, introduzido por Vaswani et al. (2017), utiliza mecanismos de atenção para processar dados sequenciais de maneira mais eficiente do que os modelos tradicionais, como *RNNs* (Redes Neurais Recorrentes) e *LSTMs* (Long Short-Term Memory).

O *GPT* aproveita essa arquitetura para criar modelos de linguagem que são inicialmente pré-treinados em grandes volumes de texto e, posteriormente, ajustados para tarefas específicas. A versão mais recente, *GPT-4*, é composta por bilhões de parâmetros, permitindo uma compreensão mais profunda e respostas mais precisas. A arquitetura *GPT* utiliza camadas empilhadas de transformadores que incluem mecanismos de atenção auto-regressiva, permitindo que o modelo se concentre em diferentes partes do texto à medida que gera respostas.

O treinamento do ChatGPT ocorre em duas fases principais: pré-treinamento e *fine-tuning* (ajuste fino).

1. Na fase de pré-treinamento, o modelo é exposto a uma grande quantidade de texto retirado de várias fontes, como livros, artigos, websites e outros materiais textuais disponíveis publicamente. O objetivo é que o modelo aprenda a prever a próxima palavra em uma sequência de texto, uma tarefa conhecida como modelagem de linguagem. Durante esse processo, o modelo aprende padrões de sintaxe, gramática, fatos do mundo e, em certa medida, raciocínio.
2. Após o pré-treinamento, o modelo passa pela fase de *fine-tuning*, onde é treinado em um conjunto de dados mais específico e cuidadosamente curado, com a supervisão humana. Nesta etapa, os treinadores usam o feedback humano para ajustar o comportamento do modelo, garantindo que ele produza respostas seguras e úteis. Este processo inclui a utilização de técnicas de aprendizado por reforço com feedback humano (*RLHF - Reinforcement Learning with Human Feedback*), onde os humanos classificam as respostas do modelo, e essas classificações são usadas para refinar ainda mais o modelo.

O ChatGPT possui várias características e capacidades notáveis que o tornam uma ferramenta poderosa e versátil:

- **Compreensão contextual:** O modelo é capaz de entender o contexto de uma conversa e manter a coerência ao longo de múltiplas interações.
- **Geração de texto:** ChatGPT pode gerar texto em uma variedade de estilos e tons, adequando-se às necessidades do usuário, seja para fins informativos, criativos ou de suporte técnico.
- **Versatilidade linguística:** Suporta múltiplos idiomas e dialetos, permitindo interações em diferentes línguas.
- **Adaptabilidade:** Pode ser adaptado para diferentes domínios de conhecimento através de fine-tuning, permitindo aplicações específicas como atendimento ao cliente, suporte técnico, tutoria educacional, entre outros.
- **Respostas detalhadas:** Capaz de fornecer respostas detalhadas e informativas, abrangendo uma ampla gama de tópicos.
- **Interatividade:** Projetado para manter diálogos contínuos e engajadores, melhorando a experiência do usuário.

Essas características fazem do ChatGPT uma ferramenta inovadora no campo da IA, com aplicações práticas em diversas áreas da sociedade moderna. O impacto do ChatGPT vai além da simples interação homem-máquina, estendendo-se a áreas cruciais como educação, saúde, negócios e entretenimento.

O avanço do ChatGPT representa um marco importante no desenvolvimento da inteligência artificial, evidenciando o potencial dos modelos de linguagem natural em transformar diversas facetas da sociedade. Com o contínuo aprimoramento e adaptação desses modelos, espera-se que suas aplicações se expandam ainda mais, contribuindo para uma interação mais eficiente e intuitiva entre humanos e máquinas. O futuro promete ainda mais inovações, impulsionadas pelo poder transformador da IA.

Neste capítulo, exploramos a tecnologia por trás do ChatGPT, um exemplo de inteligência artificial aplicada ao processamento de linguagem natural. Discutimos como o ChatGPT, fundamentado na arquitetura Transformer, representa um avanço significativo no campo da IA, permitindo interações humanas com tecnologias de IA mais naturais e eficazes.

Também discutimos que a arquitetura *Transformer*, com sua atenção, mudou como modelos de linguagem processam e geram texto, superando limitações de modelos anteriores como *RNNs* e *LSTMs*. Usando aprendizado supervisionado e não supervisionado, com pré-treinamento e *fine-tuning*, o ChatGPT pode aprender com muitos dados textuais e se adaptar a tarefas específicas, tornando-se um modelo versátil e eficiente.

O impacto do ChatGPT pode ser observado em áreas como educação, saúde, negócios e entretenimento, demonstrando o potencial da IA na sociedade. À medida que

exploramos e aprimoramos essas tecnologias, é crucial considerar as oportunidades e os desafios que essa tecnologia pode trazer em futuro breve.

Compreender as tecnologias subjacentes ao ChatGPT e outros modelos de IA é essencial para apreciar o processo de desenvolvimento de software. No próximo capítulo, exploraremos as metodologias e práticas que orientam a criação de sistemas de software robustos e eficientes. Discutiremos desde as fases iniciais de planejamento e análise de requisitos até o design, implementação, testes e manutenção de software, fornecendo uma visão de como os princípios de engenharia de software são aplicados para construir soluções tecnológicas.

2.2 DESENVOLVIMENTO DE SOFTWARE

Software refere-se a todos os componentes não físicos de um computador, rede de computadores ou dispositivo móvel. Refere-se aos programas e aplicativos (como o sistema operacional) que fazem o computador funcionar para o usuário. Software é uma coleção de instruções, dados ou programas usados para operar computadores e executar determinadas tarefas (COUTINHO; BEZERRA, 2021).

O oposto de hardware, que descreve os aspectos físicos de um computador. Em outras palavras, é um termo genérico que se refere a aplicativos, scripts e programas executados em um dispositivo. Dessa forma, é possível pensar no software como um componente variável de um computador, enquanto o hardware é constante (SAKURAI; ZUCHI, 2018).

Para Maynard (2015) os dois principais tipos de software são: software aplicativo e software de sistema. Um aplicativo é um software que atende a uma necessidade específica ou executa tarefas. Segundo Schwab (2019), o software do sistema é essencial para a operação do hardware de um computador e serve como plataforma para a execução de aplicativos. Além disso, existem outros tipos de software, como os de programação, que fornecem ferramentas necessárias para os desenvolvedores, o middleware, que atua entre o software do sistema e os aplicativos, e os drivers, que operam dispositivos e periféricos de computador.

Portanto, a interação entre diferentes tipos de software e hardware é fundamental para o funcionamento eficiente de sistemas computacionais. Enquanto o software aplicativo permite que os usuários realizem tarefas específicas, o software de sistema assegura que o hardware opere corretamente e suporte esses aplicativos. A complexidade crescente dos sistemas modernos exige um middleware eficaz para garantir que diferentes softwares funcionem harmoniosamente. Além disso, os drivers são indispensáveis para a comunicação entre o sistema e os periféricos, ampliando as funcionalidades do computador e melhorando a experiência do usuário.

Dessa forma, os componentes técnicos de um computador são controlados por software. Ao contrário do hardware de um computador, rede ou dispositivo móvel, ele não é fisicamente tangível. [Cândido \(2022\)](#) nos diz que o sistema operacional é o software fundamental de um computador ou dispositivo móvel. Em termos de interface do usuário, os três sistemas operacionais mais comuns, Windows, macOS e Linux, diferem. Outros programas podem complementar as funções básicas do sistema operacional. Eles são instalados no computador por meio de um driver ou após o download. Esses programas incluem, por exemplo, o MediaPlayer e a suíte Office da Microsoft ([NETO, 2019](#)).

Além dos sistemas operacionais, há uma variedade de softwares utilitários que desempenham funções específicas para otimizar o desempenho e a segurança do sistema. Esses utilitários incluem programas de antivírus, ferramentas de backup, e software de manutenção de disco, que ajudam a proteger os dados e a manter o computador funcionando de maneira eficiente. Além disso, com o avanço da tecnologia, o desenvolvimento de softwares de virtualização tem permitido a execução de múltiplos sistemas operacionais em uma única máquina física, aumentando a flexibilidade e a eficiência no uso de recursos de hardware.

Em certos dispositivos, o software e o hardware estão permanentemente conectados, como o computador do painel de um carro ou uma máquina de lavar. Os sistemas de navegação e os leitores de BluRay também são controlados por software, embora não possam ser substituídos por software alternativo. [Coutinho e Bezerra \(2021\)](#) nos informa que essas combinações fixas são conhecidas como "embutidos". No entanto, na década de 1950, software e hardware ainda estavam inextricavelmente ligados como uma unidade. O software era visto apenas como um componente do hardware e era referido como código de programa. John W. Tukey foi um estatístico que cunhou o termo "software" pela primeira vez em 1958 ([MAYNARD, 2015](#)). De acordo com [Sakurai e Zuchi \(2018\)](#) quase 20 anos depois, uma decisão do governo americano exigiu que a International Business Machines Corporation (IBM) listasse e calculasse seus componentes de software e hardware separadamente. Essa medida incentivou o surgimento de empresas dedicadas exclusivamente à produção de software.

Com a evolução da tecnologia, a separação entre software e hardware permitiu uma maior flexibilidade e inovação no desenvolvimento de sistemas computacionais. Essa dissociação possibilitou que o software fosse atualizado independentemente do hardware, promovendo um ciclo de inovação mais rápido. Além disso, a especialização na produção de software levou ao surgimento de uma indústria robusta, com empresas focadas em criar soluções personalizadas para diversas necessidades e setores. Isso não apenas ampliou as possibilidades de uso dos dispositivos, mas também fomentou a criação de novos modelos de negócios baseados em software, que hoje desempenham um papel crucial na economia digital global.

[Schwab \(2019\)](#) no seu livro informa que essencialmente, um programador traduz

requisitos e algoritmos em uma linguagem de programação desejada que um computador pode executar. Aplicativos, jogos e controladores de robôs são exemplos de software que podem ser criados por programação. Com a invenção de uma máquina de tear "programável", o francês Joseph-Marie Jacquard deu uma contribuição significativa para a revolução industrial. O rasgo real carecia de um processador e, em vez disso, era controlado por uma fita perfurada. Além disso, [Maynard \(2015\)](#) complementa que estes não receberam um valor entre 0 e 1, mas sim entre furo e nenhum furo. Um furo significava subir ao céu, enquanto nenhum furo significava subir ao chão. A invenção de Joseph-Maria Jacquard lançou as bases para a automação moderna.

Ainda em seu artigo [Maynard \(2015\)](#) no diz que, o próximo modelo foi Augusta Ada King-Noel, mais conhecida como Ada Lovelace, em 1843. Ela colaborou com Charles Babbage no Analytical Engine, uma calculadora mecânica de uso geral que nunca foi concluída. Ela fez várias anotações enquanto escrevia e descreveu um método para calcular os números de Bernoulli usando um computador. [Coutinho e Bezerra \(2021\)](#) complementa a ideia que, este ainda é considerado o primeiro programa de computador atual. Com o desenvolvimento do Z3 em 1941, Konrad Ernst Otto Zuse demonstrou o primeiro sistema de computador totalmente automatizado e livremente programável do mundo.

O Z3 foi desenvolvido com base nas ideias de Babbage, e sua invenção permitia realizar uma adição em menos de um quarto de segundo. Na verdade, a principal motivação para criar uma "calculadora automática" era a eficiência nos cálculos. Inicialmente, os softwares eram projetados para computadores específicos e comercializados junto com o hardware correspondente. Na década de 1980, o software começou a ser distribuído em disquetes e, posteriormente, em CDs e DVDs. Hoje em dia, a maioria dos softwares é adquirida e baixada diretamente da Internet, seja nos sites dos fornecedores ou nos sites dos provedores de serviços de aplicativos.

Assim, os modelos de ciclo de vida de software detalham as etapas que o software atravessa desde a concepção até a desativação, especificando a sequência em que essas etapas devem ocorrer. Cada fase gera produtos ou resultados que são essenciais para o desenvolvimento da próxima etapa do ciclo de vida. O principal objetivo do ciclo de vida de um software é alcançar a máxima qualidade com o menor custo e no menor tempo possível. Esse ciclo abrange diversas fases, como análise de requisitos, design, implementação, teste, lançamento, manutenção e suporte. Normalmente, um ciclo de vida de desenvolvimento de software compreende seis estágios distintos, que são os seguintes:

- Analisar Requisitos: após coletar os requisitos, eles são analisados e examinados para ver como esses dados podem ser utilizados no desenvolvimento do sistema. Após o preenchimento dos requisitos, é criado o documento Especificação de Requisitos de Software;

- **Design:** a equipe de desenvolvimento de software decide sobre os requisitos, a aparência e o cronograma de desenvolvimento do projeto. Além disso, esta fase estabelece fluxos de trabalho nos quais os engenheiros escolhem o tipo de banco de dados a ser usado para criar um modelo de dados;
- **Implementação/Codificação:** este é frequentemente o estágio mais longo do ciclo de desenvolvimento de software. Os desenvolvedores de software escrevem código baseado no documento de especificação de design do sistema. Antes de escrever a primeira linha de código, muito planejamento é necessário. As tarefas de desenvolvimento são divididas durante esta fase para que o projeto possa ser dividido em partes individuais. Em cada etapa, possui um tempo para revisar e realizar sessões de feedback para garantir que o desenvolvimento esteja no caminho certo;
- **Teste:** nesta fase, todos os tipos de testes são realizados, como testes de unidade, testes de sistema, testes de aceitação e testes de integração;
- **Manutenção:** à medida que os clientes começam a usar o sistema desenvolvido, problemas reais surgem e devem ser resolvidos caso a caso. Essa prática de cuidar de um produto em desenvolvimento é conhecida como manutenção;

Em geral, em sua pesquisa [Cândido \(2022\)](#) diz que há uma tendência de desprofissionalização nos últimos anos: cada vez mais, as pessoas estão aprendendo a programar não por meio de um curso ou treinamento formal, mas por meio de programação individualizada ou uma variedade de tutoriais online. Networking global, uma variedade de plataformas e outras oportunidades de networking estão formando grandes comunidades de entusiastas e programadores profissionais, com os limites se tornando cada vez mais tênues.

Simultaneamente, observou-se uma melhoria nos padrões e na qualidade do desenvolvimento de software: os sistemas estão se tornando cada vez mais avançados e complexos, o que leva a um aumento na demanda por sistemas de software. Isso reflete uma sociedade dinâmica e acelerada, onde as necessidades evoluem constantemente, resultando em uma maior demanda por desenvolvedores full-stack em comparação com especialistas em uma única área.

A crescente complexidade dos sistemas de software exige profissionais que possam trabalhar em todas as camadas do desenvolvimento, desde o frontend até o backend, incluindo a infraestrutura subjacente. Desenvolvedores full-stack são altamente valorizados porque possuem a flexibilidade e a competência para adaptar-se rapidamente às mudanças nas demandas do mercado e nas necessidades dos usuários. Além disso, a capacidade de um desenvolvedor full-stack de entender e integrar diferentes componentes do sistema pode levar a um desenvolvimento mais coeso e eficiente, facilitando a comunicação e a colaboração dentro das equipes de desenvolvimento. Em um ambiente onde a agilidade e a capacidade de

adaptação são cruciais, a versatilidade desses profissionais se torna um ativo indispensável para as empresas que desejam permanecer competitivas.

No entanto, o termo "desenvolvimento de software" refere-se a uma coleção de atividades de ciência da computação dedicadas à criação, design, implementação e suporte de software. As instruções ou programas que dizem a um computador o que executar são descritos pelo software. É independente de hardware e torna os computadores programáveis (SCHWAB, 2019). Maynard (2015) diz que existem três tipos básicos:

- Software de sistema para funções essenciais, incluindo sistemas operacionais, gerenciamento de disco, utilitários, gerenciamento de hardware e outras necessidades para operação;
- O software de programação fornece aos programadores ferramentas como editores de texto, compiladores, vinculadores, depuradores e outras ferramentas que geram código;
- Software aplicativo (aplicativos ou aplicativos) que auxilia os usuários na conclusão de tarefas. Exemplos incluem escrever software de produtividade, software de gerenciamento de dados e software de segurança. Os aplicativos também podem se referir a aplicativos móveis e online, como os usados para fazer compras na Amazon, compartilhar conteúdo no Facebook ou fazer upload de imagens no Instagram.

Outro tipo de software é o destinado a sistemas embarcados, que são utilizados para controlar máquinas e dispositivos que frequentemente não são identificados como computadores, como redes telefônicas, automóveis e robôs industriais. Esses dispositivos, juntamente com seus softwares, podem ser interligados no contexto da Internet das Coisas (IoT). Nesse cenário, programadores, engenheiros de software e desenvolvedores de software desempenham papéis cruciais no desenvolvimento desses sistemas. Suas responsabilidades muitas vezes se sobrepõem, e a dinâmica colaborativa pode variar conforme o departamento e a comunidade em que estão inseridos.

As soluções de engenharia de software são fundamentadas em métodos científicos e precisam funcionar efetivamente na prática, assim como pontes ou elevadores. A responsabilidade dos engenheiros de software aumentou à medida que os produtos se tornaram mais inteligentes, integrando microprocessadores, sensores e software. Hoje em dia, os produtos não se destacam mais no mercado apenas pelo software que utilizam.

De acordo com Cândido (2022), o desenvolvimento de software deve ser coordenado com o desenvolvimento de produtos mecânicos e elétricos, resultando em um papel menos definido para os desenvolvedores de software em comparação com os engenheiros. Esses desenvolvedores podem se concentrar em áreas específicas do projeto, incluindo a escrita de

código. Além disso, o ciclo de vida do desenvolvimento de software é estabelecido e envolve a conversão de requisitos em funcionalidades, o trabalho em equipe multifuncional, a gestão de equipes e processos de desenvolvimento, bem como o teste e a manutenção do software.

Segundo Schwab (2019), o papel dos desenvolvedores de software vai além das equipes de programação ou desenvolvimento, englobando também profissionais como cientistas e fabricantes de dispositivos e hardware, que contribuem para a criação de códigos de software, embora não sejam os principais responsáveis pelo desenvolvimento. Além disso, essa atuação não se restringe às indústrias tradicionais de tecnologia da informação, como empresas de software ou fabricantes de semicondutores.

Essa ampliação do escopo de atuação dos desenvolvedores de software reflete a crescente interseção entre a tecnologia da informação e outros campos, como a ciência, a medicina, a engenharia e até mesmo as artes. Hoje em dia, o desenvolvimento de software é essencial para uma variedade de indústrias, desde a automotiva até a de entretenimento, impulsionando a inovação e a transformação digital em todos os setores. Por exemplo, na indústria automotiva, os desenvolvedores de software desempenham um papel crucial no desenvolvimento de sistemas de condução autônoma e na integração de tecnologias de segurança avançadas. Da mesma forma, na indústria cinematográfica, os efeitos visuais e a animação por computador dependem fortemente do desenvolvimento de software para criar mundos virtuais e personagens digitais realistas. Assim, os desenvolvedores de software desempenham um papel cada vez mais diversificado e impactante em nossa sociedade, moldando o futuro em todas as áreas do conhecimento e da atividade humana.

2.2.1 PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

De acordo com Osterweil (2011), o desenvolvimento de um programa ou software específico para atender às necessidades de um cliente ou alcançar um determinado objetivo é apenas um dos vários produtos que surgem de uma série de processos ou atividades a serem realizados. Nesse contexto, os principais objetivos do processo incluem aumentar a conscientização sobre o assunto em questão e estabelecer canais eficazes de comunicação entre os participantes do projeto, além de garantir a manutenção do sistema e obter feedback dos usuários, conforme destacado por Silva (2020). Com isso, os objetivos principais são:

- Possuir um guia para o trabalho que é necessário e o que o cliente precisa;
- Determinar o custo de desenvolvimento;
- Manter um nível de qualidade;
- Prever o tempo de desenvolvimento.

Assim, o procedimento de criação de software representa a metodologia utilizada para construir programas de computador de qualquer natureza. Os métodos de desenvolvimento de software podem ser consideravelmente detalhados e intrincados devido às amplas aplicações atuais, abrangendo desde jogos até processadores de texto e programas de design. Essas práticas e tarefas englobam todos os elementos necessários para o desenvolvimento de software, a coordenação de equipes de desenvolvimento e a execução das diversas etapas do processo.

De acordo com [Goes et al. \(2022\)](#), embora os aplicativos possam variar em sofisticação, implementar um processo de desenvolvimento ainda é vantajoso, pois aumenta as probabilidades de sucesso. Assim, diversos métodos de desenvolvimento continuam em prática atualmente, enquanto outros, do passado, já se encontram obsoletos.

[Goes et al. \(2022\)](#) destacam que, no decorrer do projeto, partes da programação são finalizadas conforme o avanço, com novas iterações ocorrendo no meio do processo, mantendo altos níveis de exigência e testes rigorosos. Ao término do processo, a maioria das disciplinas de teste e implementação já foi concluída, embora algumas ainda possam estar em desenvolvimento e continuar necessárias. Em resumo, o tempo é dedicado à gestão do projeto em todas as fases e iterações.

Portanto, torna-se evidente que esse processo não segue o modelo cascata; em vez disso, as etapas ocorrem de forma sequencial, uma após a outra. Em vez disso, após cada iteração, podem ser realizadas entregas parciais, permitindo que os clientes avaliem se o projeto está progredindo conforme o planejado. Conforme o projeto avança, torna-se mais fácil avaliar o seu estado e identificar as próximas ações necessárias, possibilitando ajustes mais precisos no cronograma e no orçamento global.

De acordo com [Osterweil \(2011\)](#), um modelo de processo para o desenvolvimento de software, em resumo, é uma adaptação de modelos de processos existentes para a criação de aplicativos profissionais ("tipo engenheiro"). O objetivo desse modelo é simplificar a complexidade envolvida no desenvolvimento de software. Os engenheiros de software adotam um plano de desenvolvimento, já que alguns modelos de processo são desafiadores de serem implementados e mantidos.

Nessa situação, o processo de desenvolvimento é fragmentado em períodos delimitados de tempo e escopo por meio dessa abordagem (modelo procedural). O desenvolvimento do software avança por meio de etapas distintas. O gerenciamento de projetos e o controle de qualidade oferecem suporte ao desenvolvimento efetivo do software. As atividades individuais são distribuídas em várias fases do processo de desenvolvimento, e os modelos de processo podem ser aplicados uma vez (como o modelo cascata) ou de forma iterativa (como o modelo em espiral), frequentemente com ajustes sutis.

Diversas iterações conduzem ao aprimoramento iterativo (ou repetitivo) dos diversos

elementos do programa. Existem algumas discordâncias em relação aos modelos de processo de grande escala. De modo geral, o processo de desenvolvimento envolve pelo menos dois conjuntos abrangentes de atividades: a implementação de EDP (design e programação) de um lado e a análise dos processos de negócios (modelo de processo de negócios e modelo de dados) do outro.

Conforme observado por [Medeiros \(2019\)](#), a agilidade no desenvolvimento de software refere-se a abordagens que capacitam os desenvolvedores a trabalhar de maneira criativa, enquanto adiam as responsabilidades administrativas. Por sua vez, tecnologias alternativas de software, como aplicação universal e fábrica de software, questionam profundamente a metodologia tradicional de design e desenvolvimento de software. Elas adaptam software universal pronto para uso a cada conjunto de necessidades por meio de configurações. Os modelos de processo são categorizados em três tipos, como mencionado por [Silva \(2020\)](#).

O modelo inicial compreende os processos de desenvolvimento de software aplicados para supervisionar a concepção do software desde sua fase inicial até sua utilização em tempo real, abrangendo também as modificações realizadas enquanto o software está em operação.

Conforme observado por [Osterweil \(2011\)](#), a segunda abordagem diz respeito à gestão da vida útil do programa, estendendo as etapas ao longo de todo o ciclo de vida do software. O modelo de processo, que detalha os procedimentos individuais realizados tecnicamente em Tecnologia da Informação, estabelece os requisitos operacionais (ou "O QUÊ", o "COMO"). Esse tipo de modelo combina uma descrição do mundo real com uma especificação normativa. Dependendo do nível de padronização, diferentes fases de desenvolvimento são atribuídas. Algumas organizações podem certificar essas fases de evolução para empresas.

Adicionalmente, conforme argumentado pelos defensores, a abordagem no desenvolvimento de software está alinhada com a perspectiva do programador, representando um método específico de elaboração de software. Essas doutrinas muitas vezes incorporam elementos procedurais, resultando em sua categorização como modelos de processo. Consequentemente, o processo de desenvolvimento de software pode transcorrer de forma fluida ou enfrentar desafios significativos. A estratégia adotada influenciará a frequência com que cada uma dessas situações ocorre.

Com a abordagem correta, é possível minimizar o risco de o projeto ficar estagnado e exceder o orçamento estabelecido. Antes que a equipe de desenvolvimento possa dar início ao processo de criação de software, é necessário ter um projeto arquitetônico em vigor. A arquitetura de software oferece uma metodologia para auxiliar no planejamento e na organização de um sistema. Isso engloba o ambiente operacional no qual o programa será executado, os componentes que devem ser integrados e quaisquer restrições estruturais ou funcionais que devem ser consideradas.

Em contrapartida, conforme indicado por [Chaves et al. \(2020\)](#), em um cenário de arquitetura de software bem definida, é possível que a fase de desenvolvimento se torne consideravelmente desafiadora, gerando preocupação na equipe. Isso pode levar a problemas adicionais não previstos para a equipe de desenvolvimento. Dado que a escolha ideal muitas vezes é determinada pelo objetivo do produto em questão, não há uma opção claramente superior entre esses métodos de desenvolvimento.

Nessa situação, é crucial ter uma compreensão inicial dos tipos de abordagens em consideração. Se o cenário for claro e direto, a metodologia em cascata pode ser apropriada. A abordagem em cascata pode ser vantajosa para projetos com prazos apertados e poucos desafios técnicos, pois há menos probabilidade de que o resultado final não atenda às necessidades do cliente. Além disso, como a entrega é feita de forma imediata, o tempo e os recursos financeiros investidos no desenvolvimento são reduzidos, enquanto a capacidade de resposta é aumentada.

Conforme destacado por [Wieggers e Beatty \(2013\)](#), torna-se essencial adotar uma abordagem de desenvolvimento iterativo quando o projeto se torna mais complexo, uma vez que o método cascata pode, por vezes, ser desvantajoso devido à sua estrutura altamente controlada. Ao seguir as diretrizes da abordagem cascata e concluindo uma etapa de cada vez, os problemas relacionados à especificação dos critérios só são identificados após a entrega do trabalho.

Conforme observado anteriormente, esse cenário pode resultar em clientes descontentes ou, de forma mais grave, em falhas de projetos. Ao adotar uma abordagem de desenvolvimento progressivo e manter espaço para ajustes, os métodos iterativos possibilitam modificações mais eficazes em projetos de maior complexidade. Nessas situações, podem surgir dificuldades relacionadas à interpretação ou especificação dos requisitos, como mencionado por [Osterweil \(2011\)](#).

Diversas empresas que se dedicam ao desenvolvimento de software utilizam abordagens em seus procedimentos de criação. Muitos desses empreendimentos têm vínculos com o setor de defesa dos Estados Unidos, que requer uma certificação alinhada com sua estrutura de negócios para celebrar contratos. O padrão global ISO 12207 regula a metodologia de escolha, aplicação e acompanhamento do ciclo de vida do software.

Contudo, ao longo de um extenso período, estudiosos se dedicaram a encontrar abordagens reiterativas e previsíveis que potencializassem tanto a produtividade quanto a qualidade. Certas técnicas visam a sistematização ou normalização das atividades relacionadas ao desenvolvimento de software, que, em alguns casos, podem parecer desordenadas. Outras adotam táticas de gestão de projetos durante a elaboração de softwares.

Conforme apontado por [Silva \(2020\)](#), as atividades relacionadas ao desenvolvimento de software enfrentam o risco de sofrer atrasos ou exceder os custos previstos na ausência

de uma gestão de projetos adequada. É imprescindível um gerenciamento de projetos eficaz, visto que muitos projetos de software não alcançam seus objetivos estabelecidos em termos de funcionalidade, custo ou prazo de entrega.

Captar ou examinar requisitos é uma etapa crucial no processo de criação de um produto de software. Com frequência, os clientes possuem uma visão vaga do produto final, mas carecem de compreensão sobre os objetivos que o produto deve atingir. Após a coleta dos requisitos do cliente, é imperativo conduzir uma análise abrangente do escopo de desenvolvimento, o que é denominado de especificação funcional.

Antes de iniciar qualquer empreendimento de desenvolvimento, a etapa inicial e fundamental é a fase de planejamento. Durante esse período, colaboram programadores, especialistas em marketing, analistas de mercado e, crucialmente, interações com clientes para estabelecer as especificações e funcionalidades necessárias para o software. Nessa etapa, também são examinados os riscos empresariais e estabelecidos os critérios de garantia de qualidade.

Adicionalmente, a fase de implementação envolve os desenvolvedores na criação do código-fonte do projeto, alinhando-o com as especificações do software. Durante este estágio, são realizados testes de caixa preta e caixa branca. Os testes de software representam uma etapa crucial no ciclo de desenvolvimento, com o objetivo de identificar falhas o mais cedo possível.

No processo de desenvolvimento de software, a escolha da arquitetura da aplicação é uma etapa crucial que ocorre geralmente na fase de planejamento e projeto do desenvolvimento. A arquitetura de software define a estrutura geral do sistema, incluindo componentes, suas interações e as diretrizes para seu desenvolvimento e evolução. Essa escolha é fundamental porque influencia diretamente a escalabilidade, manutenção, desempenho e segurança da aplicação.

- **Escalabilidade:** A arquitetura define como o sistema pode crescer e se adaptar ao aumento de carga e usuários.
- **Manutenção:** Uma boa arquitetura facilita a adição de novas funcionalidades e a correção de bugs.
- **Desempenho:** Afeta a eficiência do sistema em termos de tempo de resposta e uso de recursos.
- **Segurança:** Define como os dados serão protegidos e como o sistema resistirá a ataques.
- **Reusabilidade:** Promove o uso de componentes reutilizáveis, economizando tempo e esforço em projetos futuros.

Dentro das diversas arquiteturas de software conhecidas, destaca-se a arquitetura hexagonal, também chamada de Arquitetura de Portas e Adaptadores, proposta por Alistair Cockburn. Essa arquitetura enfatiza a importância de isolar a lógica central do aplicativo dos detalhes de implementação externos, como interfaces de usuário, bancos de dados e *frameworks*. A separação clara entre o núcleo do aplicativo e seus componentes externos facilita a manutenção, a testabilidade e a flexibilidade do sistema. Componentes da Arquitetura Hexagonal:

- Domínio central: Contém a lógica de negócios essencial da aplicação, independente de qualquer tecnologia externa.
- Portas: Interfaces que definem como o domínio central interage com o mundo exterior. Existem dois tipos de portas:
 - Portas de entrada: Interfaces usadas pelos adaptadores de entrada para interagir com a lógica de negócios (por exemplo, serviços ou controladores).
 - Portas de saída: Interfaces que a lógica de negócios usa para interagir com sistemas externos (por exemplo, repositórios ou gateways).
- Adaptadores: Implementações concretas que fazem a ponte entre os portas e os sistemas externos. Existem dois tipos de adaptadores:
 - Adaptadores de entrada: Conectam o mundo externo ao domínio central (por exemplo, *APIs REST*, interfaces gráficas).
 - Adaptadores de saída: Conectam o domínio central a sistemas externos (por exemplo, bancos de dados, serviços de mensageria).

Essa arquitetura promove a flexibilidade e testabilidade, pois permite que a lógica de negócios seja testada isoladamente de seus componentes externos.

É importante também entender que uma API (Interface de Programação de Aplicações) é um conjunto de definições e protocolos que permite a comunicação entre diferentes sistemas de software. As APIs especificam como as solicitações de serviços devem ser feitas, quais dados devem ser enviados e o formato das respostas. Elas são essenciais para a integração entre diferentes sistemas, permitindo que aplicativos se comuniquem de forma padronizada. Aplicações modernas são frequentemente construídas com base nesse modelo, aproveitando APIs para facilitar a interoperabilidade e a escalabilidade.

Dentro do modelo de API, existe um padrão conhecido como REST (Transferência de Estado Representacional). REST é um estilo de arquitetura para a criação de serviços web que utilizam os métodos HTTP. Um serviço é considerado RESTful se seguir os princípios REST, que incluem:

- Recursos: Tudo no serviço é tratado como um recurso, identificado por um *URI (Uniform Resource Identifier)*.
- Métodos HTTP: Uso apropriado dos métodos *HTTP (GET, POST, PUT, DELETE)* para realizar operações *CRUD (Create, Read, Update, Delete)* sobre os recursos.
- Representações: Os recursos podem ser representados em diferentes formatos, como *JSON* ou *XML*.
- *Stateless*: Cada solicitação do cliente para o servidor deve conter todas as informações necessárias para entender e processar o pedido, tornando as interações sem estado.
- Cacheabilidade: As respostas devem indicar se podem ou não ser armazenadas em cache para melhorar a eficiência.

Nesse trabalho, o processo de desenvolvimento aderente será o Scrum. Utilizando essa metodologia ágil, será possível organizar e gerenciar as tarefas de forma colaborativa e iterativa, promovendo a entrega contínua de incrementos de software. O Scrum facilita a adaptação às mudanças e o alinhamento constante com os requisitos, garantindo um desenvolvimento mais eficiente e responsivo às necessidades do projeto.

2.2.2 FERRAMENTAS DE DESENVOLVIMENTO DE SOFTWARE

As ferramentas de desenvolvimento de software desempenham um papel crucial no ciclo de vida do desenvolvimento de software, oferecendo suporte às equipes de desenvolvimento em todas as etapas, desde a concepção até a implementação e manutenção de aplicativos. Essas ferramentas abrangem uma ampla variedade de recursos e funcionalidades, destinadas a facilitar e agilizar o processo de criação de software. Desde ambientes de desenvolvimento integrados (IDEs) até sistemas de controle de versão, testes automatizados e ferramentas de análise de código, a variedade de opções disponíveis reflete a diversidade de necessidades e abordagens dentro da comunidade de desenvolvimento de software. Neste contexto, explorar as diferentes categorias e funcionalidades das ferramentas de desenvolvimento de software torna-se fundamental para entender como elas podem beneficiar os projetos de software e impulsionar a eficiência e a qualidade do desenvolvimento.

A escolha das ferramentas JIRA, Java, JUnit, Postman, Spring Boot e H2 para o desenvolvimento deste projeto é baseada em suas funcionalidades robustas e amplamente reconhecidas na indústria de software.

JIRA é uma ferramenta de gestão de projetos ágil que permite acompanhar tarefas, sprints e problemas de forma eficiente, facilitando a colaboração e a transparência dentro da equipe. Sua flexibilidade e integração com outras ferramentas o tornam ideal para gerenciar projetos complexos.

Java é uma linguagem de programação versátil e amplamente utilizada, conhecida por sua portabilidade e desempenho. Sua vasta comunidade de desenvolvedores e a abundância de bibliotecas e frameworks facilitam a resolução de problemas e a implementação de funcionalidades diversas.

JUnit é uma ferramenta essencial para testes unitários em Java, promovendo o desenvolvimento orientado a testes (TDD). Com JUnit, é possível garantir a qualidade e a robustez do código, detectando e corrigindo erros precocemente no ciclo de desenvolvimento.

Postman é uma ferramenta poderosa para testar APIs. Sua interface intuitiva permite criar, executar e automatizar testes de APIs de maneira eficiente, garantindo que os serviços web funcionem conforme o esperado.

Spring Boot é um framework que simplifica o desenvolvimento de aplicações Java baseadas em Spring. Ele permite a criação rápida e simplificada de aplicações robustas e escaláveis, com configurações mínimas e suporte a uma vasta gama de funcionalidades prontas para uso.

H2 é um banco de dados em memória que é leve e fácil de integrar com aplicações Java. Ele é ideal para desenvolvimento e testes devido à sua rápida configuração e desempenho, permitindo uma iteração rápida durante o processo de desenvolvimento.

Essas ferramentas foram escolhidas por sua sinergia e eficiência comprovada em projetos de software, proporcionando um ambiente de desenvolvimento produtivo e confiável. Optar por outras ferramentas poderia não oferecer o mesmo nível de integração, suporte comunitário e robustez necessários para atingir os objetivos do projeto com a mesma eficácia.

2.2.2.1 JIRA

O Jira é uma ferramenta de gerenciamento de projetos desenvolvida pela *Atlassian*, uma empresa de tecnologia amplamente reconhecida. Ele é amplamente utilizado no processo de desenvolvimento de software para auxiliar equipes a planejar, acompanhar e gerenciar suas atividades de forma eficiente.

Em sua essência, o Jira funciona como um sistema de rastreamento de problemas e tarefas, permitindo que as equipes organizem suas atividades em unidades gerenciáveis, como projetos, histórias, tarefas, bugs e muito mais. Ele oferece uma interface intuitiva e personalizável, adaptável às necessidades específicas de cada equipe e projeto.

Uma das principais características do Jira é sua capacidade de acompanhar o progresso do projeto em tempo real. Por meio de quadros Kanban, listas de tarefas e gráficos de progresso, os membros da equipe podem visualizar facilmente o status das atividades, identificar gargalos e tomar medidas corretivas rapidamente.

Além disso, o Jira facilita a colaboração entre membros da equipe, permitindo que eles

comentem e compartilhem informações diretamente nas tarefas. Isso promove a transparência e a comunicação eficaz, reduzindo a necessidade de reuniões desnecessárias e trocas de e-mails intermináveis. No contexto do desenvolvimento de software, o Jira pode ser usado para uma variedade de finalidades, incluindo:

- **Gestão de Requisitos:** Captura, prioriza e gerencia os requisitos do projeto, desde os mais amplos até os detalhados, garantindo que todas as partes interessadas estejam alinhadas com o que está sendo desenvolvido.
- **Planejamento de Sprints:** Permite que as equipes de desenvolvimento planejem suas iterações (sprints), atribuindo tarefas a cada membro da equipe, estimando esforços e definindo metas claras para o período.
- **Rastreamento de Problemas:** Registra e monitora bugs, problemas e solicitações de alterações, garantindo que nada seja esquecido e que todos os problemas sejam tratados de maneira oportuna.
- **Integração com Ferramentas de Desenvolvimento:** Integra-se facilmente com outras ferramentas de desenvolvimento, como Bitbucket, GitHub e Jenkins, permitindo uma integração contínua e uma visão holística do processo de desenvolvimento.
- **Relatórios e Métricas:** Fornece relatórios detalhados e métricas sobre o desempenho do projeto, permitindo que as equipes identifiquem áreas de melhoria e tomem decisões informadas.

Em resumo, o Jira é uma ferramenta poderosa que pode ser adaptada para atender às necessidades específicas de qualquer equipe de desenvolvimento de software. Ao centralizar todas as atividades do projeto em um único local e fornecer recursos avançados de acompanhamento e colaboração, ele ajuda as equipes a trabalharem de maneira mais eficiente, entregando produtos de alta qualidade dentro do prazo e do orçamento estabelecidos.

2.2.2.2 JAVA

Java é uma linguagem de programação de propósito geral que foi criada pela Sun Microsystems (adquirida posteriormente pela Oracle Corporation) na década de 1990. Desde então, tornou-se uma das linguagens mais populares e amplamente utilizadas em todo o mundo, sendo reconhecida por sua portabilidade, robustez e ampla adoção em uma variedade de domínios, desde o desenvolvimento de aplicativos para desktop até sistemas empresariais de grande escala e aplicativos móveis.

Uma das características mais marcantes do Java é sua capacidade de ser executado em diferentes plataformas sem a necessidade de recompilação. Isso é possível graças à sua

máquina virtual Java (JVM), que atua como uma camada de abstração entre o código Java e o sistema operacional subjacente. Isso significa que um programa Java compilado pode ser executado em qualquer dispositivo ou sistema que tenha uma JVM compatível instalada, tornando-o altamente portátil e interoperável.

Além de sua portabilidade, o Java é conhecido por sua sintaxe simples e fácil de aprender, tornando-o uma escolha popular para iniciantes em programação. Sua abordagem orientada a objetos fornece um modelo de desenvolvimento flexível e modular, facilitando a criação e manutenção de código limpo e escalável.

Outro aspecto fundamental do Java é sua vasta biblioteca padrão, conhecida como Java Standard Edition (Java SE), que fornece um conjunto abrangente de classes e métodos para lidar com uma ampla gama de tarefas, desde manipulação de arquivos até comunicação em rede e processamento de dados.

Além do Java SE, existem também outras edições que estendem a linguagem para fins específicos, como o Java Enterprise Edition (Java EE), voltado para o desenvolvimento de aplicativos corporativos e sistemas distribuídos, e o Java Micro Edition (Java ME), projetado para dispositivos com recursos limitados, como dispositivos móveis e embarcados.

Nos últimos anos, o ecossistema Java tem continuado a evoluir com o lançamento de novas versões e recursos, como o Java Platform Module System (JPMS) introduzido no Java 9, que visa tornar o desenvolvimento e implantação de aplicativos Java mais modular e escalável.

Em suma, o Java é uma linguagem de programação versátil, poderosa e altamente popular, amplamente utilizada em uma variedade de cenários de desenvolvimento de software. Sua portabilidade, simplicidade, robustez e vasta comunidade de desenvolvedores contribuíram para sua posição como uma das linguagens mais influentes e amplamente adotadas na indústria de tecnologia.

2.2.2.3 JUNIT

O JUnit é um framework de teste unitário para a linguagem de programação Java. Ele oferece uma estrutura simples e eficaz para escrever e executar testes automatizados em código Java, ajudando os desenvolvedores a garantir que suas classes e métodos funcionem conforme o esperado.

Desenvolvido por Erich Gamma e Kent Beck, o JUnit é uma das ferramentas de teste mais amplamente utilizadas na comunidade Java devido à sua simplicidade e poder. Ele segue o padrão xUnit, que define um conjunto de regras e convenções para estruturar e executar testes unitários em diferentes linguagens de programação.

Com o JUnit, os desenvolvedores podem criar classes de teste que verificam o

comportamento de unidades individuais de código, como métodos de uma classe. Eles podem definir casos de teste para validar diferentes cenários de entrada e saída, garantindo que o código produza os resultados esperados em uma variedade de situações.

O JUnit fornece uma série de anotações e assertivas que facilitam a escrita de testes claros e concisos. As anotações são usadas para marcar métodos como casos de teste ou configurar o ambiente de teste, enquanto as assertivas são usadas para verificar se determinadas condições são verdadeiras durante a execução do teste.

Além disso, o JUnit oferece integração com diversas ferramentas de desenvolvimento e build, como Maven e Gradle, permitindo que os testes sejam facilmente incorporados ao fluxo de desenvolvimento de software. Isso ajuda a garantir que os testes sejam executados regularmente durante o ciclo de desenvolvimento, proporcionando feedback rápido sobre possíveis problemas no código.

Em resumo, o JUnit é uma ferramenta essencial para desenvolvedores Java que desejam garantir a qualidade e a confiabilidade de seu código por meio de testes automatizados. Sua simplicidade, poder e integração com outras ferramentas de desenvolvimento o tornam uma escolha popular para testes unitários em uma variedade de projetos de software.

2.2.2.4 POSTMAN

O Postman é uma ferramenta amplamente utilizada por desenvolvedores de software para testar, documentar e colaborar em APIs. É uma aplicação de desktop que oferece uma interface fácil de usar para realizar uma variedade de tarefas relacionadas a APIs.

Com o Postman, os desenvolvedores podem enviar solicitações HTTP para APIs e receber respostas, facilitando o teste e a depuração de serviços web. Ele oferece suporte a vários métodos HTTP, como GET, POST, PUT e DELETE, permitindo que os usuários simulem diferentes tipos de interações com a API.

Além de testar APIs, o Postman também é útil para documentá-las. Os desenvolvedores podem criar coleções de solicitações, adicionar descrições e exemplos, e compartilhar facilmente essa documentação com outros membros da equipe ou com a comunidade em geral.

Uma característica poderosa do Postman é sua capacidade de automatizar tarefas repetitivas por meio de scripts. Os usuários podem escrever scripts usando JavaScript para pré-processar solicitações, extrair dados de respostas e realizar validações, tornando o processo de teste mais eficiente e confiável.

Além disso, o Postman oferece recursos de colaboração que permitem que os membros da equipe trabalhem juntos em projetos de API. Eles podem compartilhar coleções, sincronizar alterações e revisar solicitações, promovendo uma abordagem colaborativa para

o desenvolvimento de APIs.

Em resumo, o Postman é uma ferramenta essencial para desenvolvedores que trabalham com APIs, oferecendo uma maneira fácil e eficaz de testar, documentar e colaborar em serviços web. Sua interface intuitiva, recursos de automação e capacidades de colaboração o tornam uma escolha popular entre equipes de desenvolvimento em todo o mundo.

2.2.2.5 SPRING BOOT

O *Spring Boot* é um *framework* de desenvolvimento de aplicativos Java que oferece uma abordagem simplificada e eficiente para criar aplicativos robustos e escaláveis. Desenvolvido pela Pivotal Software (agora parte da VMware), o *Spring Boot* baseia-se no ecossistema *Spring Framework*, amplamente utilizado na indústria de desenvolvimento de software.

Uma das características mais marcantes do *Spring Boot* é sua capacidade de simplificar o processo de configuração e inicialização de aplicativos Java. Ele adota uma abordagem de "convenção sobre configuração", o que significa que muitas configurações padrão são pré-configuradas automaticamente, permitindo que os desenvolvedores se concentrem na lógica de negócios do aplicativo.

O *Spring Boot* também oferece um conjunto abrangente de ferramentas e bibliotecas que facilitam o desenvolvimento de aplicativos. Ele fornece suporte integrado para a criação de APIs *RESTful*, integração com bancos de dados, autenticação e autorização, entre outros recursos comuns de aplicativos corporativos.

Além disso, o *Spring Boot* é altamente modular e extensível, o que significa que os desenvolvedores podem escolher os componentes específicos que desejam usar em seus aplicativos e facilmente adicionar ou remover funcionalidades conforme necessário.

Outra vantagem do *Spring Boot* é sua forte integração com outras tecnologias do ecossistema Spring, como *Spring Data*, *Spring Security* e *Spring Cloud*. Isso permite que os desenvolvedores construam aplicativos complexos e distribuídos de forma coesa e coesa, aproveitando as melhores práticas e padrões de design estabelecidos pelo Spring Framework.

Em resumo, o *Spring Boot* é uma ferramenta poderosa para o desenvolvimento de aplicativos Java, oferecendo uma maneira simples e eficaz de criar aplicativos robustos, escaláveis e de alta qualidade. Sua abordagem de "opinião sobre configuração", ampla gama de recursos e integração com outras tecnologias do ecossistema Spring o tornam uma escolha popular entre desenvolvedores em todo o mundo.

2.2.2.6 H2

O H2 é um Sistema de Gerenciamento de Banco de Dados relacional (SGBDR) escrito em Java. Uma de suas características mais distintivas é o suporte para banco de dados em memória. Esse recurso permite que os desenvolvedores criem bancos de dados temporários que residem inteiramente na memória do computador, em oposição aos tradicionais bancos de dados que são armazenados em disco.

Esses bancos de dados em memória são extremamente úteis para aplicações que precisam de alta performance e escalabilidade, especialmente em ambientes de desenvolvimento e teste, onde os dados podem ser facilmente descartados e recriados sem afetar o estado do sistema.

Além disso, o H2 oferece suporte completo à linguagem SQL padrão e possui uma ampla gama de recursos, incluindo suporte a transações ACID (Atomicidade, Consistência, Isolamento, Durabilidade), índices, consultas complexas e procedimentos armazenados. Isso o torna uma escolha popular para uma variedade de aplicativos, desde aplicativos da web até aplicativos corporativos de missão crítica.

Outra característica interessante do H2 é a sua capacidade de ser integrado diretamente em aplicativos Java. Ele pode ser incorporado como uma biblioteca dentro do aplicativo, eliminando a necessidade de configurar e gerenciar um servidor de banco de dados separado.

No entanto, é importante notar que, devido à natureza volátil dos bancos de dados em memória, os dados armazenados neles são perdidos quando o aplicativo é encerrado. Portanto, eles são mais adequados para casos de uso onde a persistência de dados não é uma preocupação primordial, como testes automatizados, prototipagem rápida e armazenamento temporário de dados.

Em resumo, o H2 banco de dados em memória é uma ferramenta poderosa e versátil que oferece uma solução eficaz para casos de uso que exigem alta performance e escalabilidade. Sua facilidade de uso, ampla gama de recursos e integração perfeita com aplicativos Java o tornam uma escolha popular entre desenvolvedores em todo o mundo.

2.2.3 TIPOS DE DESENVOLVIMENTO DE SOFTWARE

O campo do desenvolvimento de software é vasto e diversificado, abrangendo uma ampla gama de abordagens, metodologias e práticas. Compreender os diferentes tipos de desenvolvimento de software é essencial para os profissionais da área, pois cada tipo apresenta características específicas que se adequam a diferentes contextos e necessidades do projeto. Desde abordagens tradicionais, como o modelo em cascata, até metodologias ágeis, como Scrum e Kanban, e técnicas emergentes, como DevOps e desenvolvimento orientado a microsserviços, há uma variedade de opções disponíveis para equipes de desenvolvimento.

Explorar os diferentes tipos de desenvolvimento de software não apenas oferece insights sobre as melhores práticas e estratégias, mas também ajuda a escolher a abordagem mais adequada para alcançar os objetivos do projeto de forma eficaz e eficiente.

2.2.4 METODOLOGIAS ÁGEIS APLICADAS NO DESENVOLVIMENTO DE SOFTWARE

As metodologias ágeis aplicadas no desenvolvimento de software representam uma abordagem dinâmica e adaptável para a criação de produtos de software. Essas metodologias surgiram como uma resposta às limitações das abordagens tradicionais de desenvolvimento de software, como o modelo em cascata, que muitas vezes eram inflexíveis e pouco eficazes na gestão de projetos complexos. As metodologias ágeis valorizam a colaboração em equipe, a comunicação contínua com os clientes e a capacidade de resposta a mudanças ao longo do ciclo de vida do projeto. Elas se baseiam em princípios como a entrega incremental de software funcional, a priorização de requisitos com base no valor do negócio e a realização de ciclos curtos de desenvolvimento conhecidos como "sprints". Ao adotar metodologias ágeis, as equipes de desenvolvimento podem aumentar sua flexibilidade, melhorar a qualidade do produto final e reduzir o tempo de entrega, permitindo uma maior adaptação às necessidades em constante mudança do mercado.

De acordo com [Sanches \(2019\)](#), as metodologias ágeis de desenvolvimento de software visam a entrega rápida de pequenos incrementos funcionais para aumentar a satisfação do cliente. Priorizando a melhoria contínua, essas abordagens empregam métodos adaptáveis e colaborativos. Equipes auto-organizadas e de pequeno porte, compostas por engenheiros de software e stakeholders de negócios, frequentemente interagem pessoalmente ao longo de todo o ciclo de vida do desenvolvimento, como parte da prática ágil de desenvolvimento de software.

Segundo a pesquisa de [Pontes e Arthaud \(2018\)](#), um grupo de engenheiros de software formulou um conjunto de estratégias de desenvolvimento ágil em resposta às limitações percebidas do modelo de desenvolvimento em cascata, que divide os projetos em várias fases lineares consecutivas. Em 2001, esses profissionais se reuniram em Utah para investigar essas abordagens de desenvolvimento em menor escala. Posteriormente, eles publicaram o Manifesto Ágil de Desenvolvimento de Software, que estabelece um conjunto de princípios para o desenvolvimento de software flexível e iterativo.

Esses princípios enfatizam a instrução, o trabalho em equipe, a capacidade de resposta e o desenvolvimento de soluções de software funcionais em vez de produtos finais que podem ser atualizados posteriormente. Graças à colaboração de todos os membros do projeto, a abordagem ágil representa uma estratégia de gestão de projetos centrada no cliente e no produto nessa situação específica. Ele serve como um guia para a gestão de tarefas e a realização de projetos, permanecendo adaptável conforme necessário e conforme

as exigências do cliente evoluem durante a criação de um produto.

[Goes et al. \(2022\)](#) sustenta a ideia de que as metodologias ágeis têm como objetivo principal adaptar-se prontamente às mudanças enquanto produzem software funcional. No entanto, cada uma dessas abordagens descreve os estágios de desenvolvimento de software de maneira distinta. Metodologias ágeis no desenvolvimento de software incluem: Scrum, Lean Software Development (LSD), Extreme Programming, Crystal, Kanban, Método de Desenvolvimento de Sistemas Dinâmicos e Desenvolvimento Orientado a Recursos.

Scrum é uma estrutura ágil simplificada que os gerentes de projeto podem utilizar para gerenciar projetos de qualquer escala, sejam eles iterativos ou incrementais. No Scrum, o proprietário do produto cria um backlog do produto, permitindo que ele colabore com sua equipe na identificação e priorização das funcionalidades do sistema. O backlog do produto inclui correções de bugs, funcionalidades e requisitos não funcionais. Após a definição do backlog do produto, apenas a equipe associada pode adicionar novos itens.

Como discutido por [Silva, Santos e Shibao \(2019\)](#), após a equipe e o proprietário do produto concordarem com os objetivos, as equipes multifuncionais se comprometem a entregar incrementos de software funcional ao final de cada sprint, geralmente em 30 dias. O backlog do produto é revisado, investigado e priorizado após cada sprint, selecionando um novo conjunto de tarefas para o sprint seguinte. A popularidade do Scrum aumentou ao longo do tempo devido à sua simplicidade, produtividade e à capacidade de incorporar os diversos princípios recomendados por outras metodologias ágeis.

A adoção do Scrum neste trabalho se justifica pela sua capacidade de promover a colaboração e a flexibilidade dentro de um projeto. Scrum, sendo uma metodologia ágil, facilita a adaptação rápida a mudanças nos requisitos do projeto, algo crucial em um ambiente dinâmico de desenvolvimento de software. A estrutura do Scrum, com sprints curtos e reuniões diárias (daily stand-ups), garante que todos os membros da equipe estejam alinhados e trabalhando de forma coordenada para atingir objetivos comuns. Além disso, as revisões e retrospectivas de sprint permitem uma melhoria contínua, ajudando a identificar e resolver problemas de forma proativa.

Outra razão para utilizar Scrum é sua ênfase na entrega incremental de funcionalidades. Em vez de esperar até o final do ciclo de desenvolvimento para entregar o produto, o Scrum permite a entrega de incrementos de software funcionais em intervalos regulares. Isso não apenas aumenta a visibilidade do progresso do projeto para os stakeholders, mas também proporciona feedback constante e valioso que pode ser utilizado para ajustar e melhorar o produto continuamente. A abordagem iterativa do Scrum, portanto, contribui significativamente para a qualidade do software e para a satisfação do cliente, tornando-o uma escolha ideal para este trabalho.

2.2.5 APLICAÇÃO DO DESENVOLVIMENTO DE SOFTWARE NO CONTEXTO DO CHATGPT

O ChatGPT pode ser integrado ao processo de desenvolvimento de software de diversas maneiras, atuando como um assistente virtual que auxilia os desenvolvedores em várias etapas do ciclo de vida do software. Desde a fase de planejamento e design até a implementação, testes e manutenção, o ChatGPT pode fornecer suporte na elaboração de documentação, na geração de código, na revisão de códigos, na resolução de problemas e até mesmo na gestão de projetos. Ferramentas de integração contínua *CI* e entrega contínua *CD* podem ser configuradas para utilizar o ChatGPT na análise de código e na recomendação de melhorias. Além disso, pode ser utilizado em chatbots internos para responder perguntas frequentes, oferecer tutoriais ou fornecer exemplos de código.

Os benefícios da utilização do ChatGPT são variados e significativos. Exemplos de caso de uso do ChatGPT no desenvolvimento de software:

- **Geração de código:** Um desenvolvedor pode utilizar o ChatGPT para gerar trechos de código em diferentes linguagens de programação, economizando tempo e reduzindo erros.
- **Revisão de código:** Equipes podem usar o ChatGPT para automatizar parte da revisão de código, identificando potenciais problemas de segurança ou performance.
- **Documentação automatizada:** O ChatGPT pode ser usado para gerar documentação de APIs, guias de usuário e manuais técnicos com base no código e nos comentários fornecidos pelos desenvolvedores.
- **Suporte a DevOps:** Pode auxiliar na criação e manutenção de scripts de automação para integração e entrega contínua.
- **Apoio à resolução de problemas:** O ChatGPT pode oferecer soluções para problemas comuns de programação ou configuração, funcionando como uma base de conhecimento interativa.

A integração do ChatGPT no desenvolvimento de software representa uma evolução significativa na maneira como os projetos são geridos e executados. Ele traz eficiência, reduz a carga de trabalho manual e melhora a qualidade do produto final. No entanto, é crucial entender suas limitações e utilizá-lo como um complemento, e não como substituto do julgamento e da criatividade humana. Sua implementação adequada pode transformar processos, tornando-os mais ágeis e responsivos às mudanças e demandas do mercado.

Neste capítulo, são explorados diversos aspectos do processo de desenvolvimento de software, incluindo a integração de metodologias ágeis, a arquitetura de software e a elucidação de requisitos. Discute-se detalhadamente como o ChatGPT pode contribuir

em cada uma dessas áreas, proporcionando um suporte benéfico e eficiente. No próximo capítulo, será desenvolvido uma aplicação para uma cafeteria, utilizando o ChatGPT em todas as etapas do processo de desenvolvimento, e será realizado análises baseadas nessa experiência.

3 Metodologia

Este capítulo apresenta uma aplicação para cafeteria desenvolvida segundo princípios sólidos de engenharia de software, como os princípios SOLID, e submetida a testes unitários para garantir ao menos funcionalidade básica e sua robustez e confiabilidade. A aplicação foi construída utilizando tecnologias dos últimos cinco anos, incluindo Java 17, Spring Boot e bancos de dados em memória H2.

Um aspecto desafiador deste projeto, foi a utilização do ChatGPT ao longo de todo o processo de desenvolvimento, desde a coleta de requisitos até a escrita de código. O objetivo deste capítulo é explorar o impacto do ChatGPT no processo de desenvolvimento, avaliando sua eficiência, a qualidade do código gerado e a conformidade com os requisitos funcionais e não funcionais da aplicação.

Ao longo deste capítulo, analisaremos como o ChatGPT impactou cada etapa do desenvolvimento e, no capítulo a seguir, destacaremos suas vantagens e limitações ao longo desse processo. Examinaremos a eficiência do desenvolvimento, a qualidade do código produzido e como o ChatGPT contribuiu para a adequação da aplicação aos requisitos. Essa análise fornecerá *insights* sobre o uso de inteligência artificial no desenvolvimento de software, apontando caminhos para futuras pesquisas e avanços na área.

Além do suporte do ChatGPT, foram criados diversos diagramas para auxiliar no entendimento geral do sistema. Esses diagramas serviram como ferramentas visuais importantes para comunicar a arquitetura, o fluxo de dados e as interações entre os componentes da aplicação, melhorando a clareza do projeto como um todo.

3.0.1 QUESTÕES DE PESQUISA

1. Analisar o funcionamento do ChatGPT como uma ferramenta no processo de desenvolvimento de software;

3.0.2 MODELAGEM DO SOFTWARE

Como etapa inicial na análise e modelagem do sistema, foram definidos os requisitos funcionais e não funcionais, o Jira será utilizado como principal ferramenta na descrição desses requisitos, seguindo o padrão Scrum, onde foram criadas user stories para todos os requisitos. Esses requisitos têm como objetivo delimitar as características e operações que o sistema deve e não deve possuir. Eles serão apresentados em imagens a seguir para fornecer uma visão clara das necessidades e expectativas em relação ao sistema em desenvolvimento.

3.0.2.1 REQUISITOS

Todos os requisitos foram colhidos com a ajuda do ChatGPT. Utilizando comandos simples, como "Crie um user story para uma aplicação de cafeteria: 'Quando o pedido estiver pronto, o cliente pode retirar a ordem, e a ordem deve ser alterada para o status de retirado'", conseguimos gerar descrições detalhadas e das funcionalidades necessárias. O ChatGPT forneceu a estrutura básica dos user stories, e também acrescentou algumas notas e observações que algumas vezes podem melhorar o entendimento e a clareza dos requisitos. O documento de requisitos criado com a ajuda do ChatGPT está disponível no [link](#).

Após a criação dos user stories pelo ChatGPT, todo o texto foi transcrito para o Jira. Durante esse processo, as observações adicionais feitas pelo ChatGPT foram integradas aos tickets no Jira, proporcionando um contexto para cada requisito.

Métricas de Sucesso

Objetivo	Métrica
O cliente deve ser capaz de fazer toda a operação de pedir um café a fazer o pagamento.	Não deverá ser retornado qualquer erro ao cliente em todo esse fluxo.

Fonte: Adaptado pelo autor

Figura 4 – User story 1

Projects / Cafeteria Puc / Add epic / CP-3

Configuração Personalizada do Pedido de Café

Attach Add a child issue Link issue Create ...

Epic <https://cafeteriatcc.atlassian.net/jira/core/projects/CP/timeline?rangeMode=weeks&selectedIssue=CP-2>

Description

O sistema deve permitir que o cliente faça um pedido de café de maneira personalizada. O cliente deverá ter a opção de selecionar especificamente o tipo de café (como expresso, latte ou cappuccino, etc.), o tipo de leite (integral, desnatado, sem lactose, vegetal, etc.), o tamanho da bebida (pequeno e grande) e a modalidade de consumo (consumir no local ou para levar).

Critérios de Aceitação:

1. O cliente deve ser capaz de escolher entre pelo menos três tipos diferentes de café.
2. O cliente deve ter a opção de selecionar entre pelo menos três tipos de leite.
3. Deve ser possível escolher entre dois tamanhos de bebida.
4. O sistema deve oferecer uma opção clara para selecionar o local de consumo: no estabelecimento ou para levar.

Fonte: Adaptado pelo autor

Figura 5 – User story 2

Projects / Cafeteria Puc / Add epic / CP-4

Adição de itens à ordem antes do pagamento

Attach Add a child issue Link issue Create ...

Epic None

Description

Como usuário, quero ter a opção de adicionar itens ao meu pedido a qualquer momento antes de realizar o pagamento, para que eu possa ajustar minha compra conforme necessário sem ter que iniciar um novo pedido.

Critérios de Aceitação:

1. **Adição de Itens:** O usuário deve ser capaz de adicionar itens adicionais a ordem de compras em qualquer etapa do processo de pedido, antes da finalização do pagamento.
2. **Confirmação de Adição:** Após adicionar um item, o usuário deve receber uma confirmação de que o item foi adicionado com sucesso ao pedido.
3. **Opção de Edição:** O usuário deve ter a opção de editar quantidades ou remover itens da ordem antes do pagamento.

Fonte: Adaptado pelo autor

Figura 6 – User story 3

Projects / Cafeteria Puc / Add epic / CP-5

O cliente pode cancelar uma ordem antes do pagamento

Attach Add a child issue Link issue Create ...

Epic None

Description

Como usuário, quero ter a opção de cancelar minha ordem a qualquer momento antes de seu processamento final, para que eu possa desistir da compra se mudar de ideia ou cometer um erro durante o pedido.

- Acesso à Opção de Cancelamento:** O usuário deve ter fácil acesso à opção de cancelar o pedido em qualquer etapa antes da finalização do processamento.
- Feedback Imediato:** Após o cancelamento, o usuário deve receber uma notificação imediata confirmando que o pedido foi cancelado.
- Registro de Cancelamento:** O cancelamento deve ser registrado no sistema para referência futura e controle administrativo.

Fonte: Adaptado pelo autor

Figura 7 – User story 4

Projects / Cafeteria Puc / Add epic / CP-6

Quando o pedido é pago, não são permitidas alterações.

Attach Add a child issue Link issue Create ...

Epic None

Description

Como usuário, desejo ser informado que não poderei fazer alterações no pedido uma vez que o pagamento seja efetuado, para garantir que eu finalize minhas escolhas corretamente antes de proceder com o pagamento.

Critérios de Aceitação:

- Notificação Pré-Pagamento:** Antes de realizar o pagamento, o usuário deve ser claramente notificado de que não poderá fazer alterações no pedido após este ponto.
- Confirmação de Pedido:** O sistema deve solicitar que o usuário revise e confirme todos os detalhes do pedido antes de proceder ao pagamento.
- Bloqueio de Alterações:** Uma vez que o pagamento é confirmado, todas as opções de edição ou cancelamento devem ser desativadas no sistema.
- Feedback ao Tentar Alterar:** Se o usuário tentar fazer alterações após o pagamento, uma mensagem deve ser exibida informando que alterações não são permitidas após o pagamento.

Fonte: Adaptado pelo autor

Figura 8 – User story 5

Projects / Cafeteria Puc / Add epic / CP-7

Pagamento de Pedido com Cartão de Crédito

Attach Add a child issue Link issue Create ...

Epic None

Description

Como usuário, quero poder pagar meu pedido utilizando um cartão de crédito para que eu possa concluir minha compra de maneira rápida e segura.

Critérios de Aceitação:

- 1. Suporte a Principais Bandeiras de Cartão:** O sistema deve aceitar as principais bandeiras de cartão de crédito como Visa, MasterCard, American Express, entre outras.
- 2. Processamento de Pagamento:** Após a inserção dos dados, o sistema deve processar o pagamento de forma eficaz e segura.
- 3. Confirmação de Pagamento:** O usuário deve receber uma confirmação imediata de que o pagamento foi realizado com sucesso.

Fonte: Adaptado pelo autor

Figura 9 – User story 6

Projects / Cafeteria Puc / Add epic / CP-8

Emissão de Recibo Após Pagamento do Pedido

Attach Add a child issue Link issue Create ...

Epic None

Description

Como cliente, quero receber um recibo após o pagamento do meu pedido para que eu possa ter um registro da transação para referência futura ou para fins de reembolso.

Critérios de Aceitação:

- 1. Disponibilidade do Recibo:** Após a confirmação do pagamento, o recibo deve estar imediatamente disponível para o cliente.
- 2. Informações Detalhadas:** O recibo deve conter detalhes completos do pedido, incluindo itens comprados, preços individuais, total pago, data da transação.

Fonte: Adaptado pelo autor

Figura 10 – User story 7

Projects / Cafeteria Puc / Add epic / CP-10

Alteração de status do pedido como pronto pelo barista

Attach Add a child issue Link issue Create ...

Epic None

Description

Como barista, quero ter a capacidade de marcar um pedido como pronto assim que terminar a preparação, para que a equipe de atendimento possa entregar imediatamente ao cliente, garantindo eficiência e satisfação do cliente.

Critérios de Aceitação:

- Atualização de Status em Tempo Real:** Ao marcar um pedido como pronto, o status do pedido deve ser atualizado em tempo real no sistema para que a equipe de atendimento e o cliente sejam imediatamente notificados.
- Registro de Atividade:** O sistema deve registrar o momento exato em que o pedido foi marcado como pronto, para controle de tempo de serviço e análise de desempenho.
- Suporte para Múltiplos Pedidos:** O barista deve ser capaz de gerenciar e marcar múltiplos pedidos como prontos simultaneamente, caso necessário.

Fonte: Adaptado pelo autor

Figura 11 – User story 8

Projects / Cafeteria Puc / Add epic / CP-11

Quando o pedido estiver pronto, o cliente pode retirar a ordem, e a ordem deve ser alterada para o status de retirado

Attach Add a child issue Link issue Create ...

Epic None

Description

Como cliente, quero poder retirar meu pedido assim que ele estiver pronto e marcar como retirado no sistema, para que o estabelecimento tenha um registro claro de que a transação foi concluída com sucesso.

Critérios de Aceitação:

- Atualização de Status em Tempo Real:** Assim que o cliente marcar o pedido como retirado, o status deve ser atualizado no sistema para "Retirado" em tempo real.
- Feedback ao Cliente:** Após marcar o pedido como retirado, o cliente deve receber uma confirmação final, agradecendo pela visita e compra.

Fonte: Adaptado pelo autor

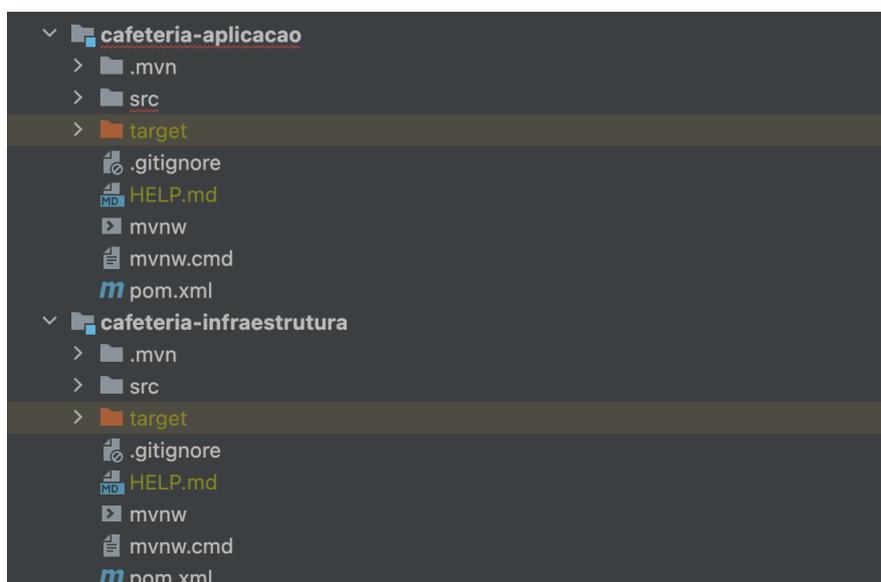
3.0.2.2 ARQUITETURA DO SOFTWARE

Neste projeto decidimos adotar a arquitetura hexagonal para estruturar nosso projeto de software. Essa abordagem nos permite criar sistemas mais robustos, modulares e facilmente testáveis, além de promover a independência das tecnologias utilizadas. Como pode ser visto na figura 12, o projeto foi dividido em dois pacotes principais, seguindo a arquitetura hexagonal. O pacote da aplicação contém todas as regras de negócio da aplicação de cafeteria, ou seja, a camada de domínio. Já o pacote de infraestrutura contém todos os componentes responsáveis por lidar com a conexão com o banco de dados e por receber requisições HTTP.

Ao tentar definir a estrutura do projeto utilizando o ChatGPT, a resposta obtida não foi tão clara e acabou sendo um pouco confusa. Esse aspecto ressalta um desafio importante: se o profissional de desenvolvimento de software não possui um conhecimento sólido, ele pode enfrentar dificuldades significativas ao tentar modelar a arquitetura do projeto.

Para mitigar esses problemas, é essencial que o profissional de desenvolvimento tenha um entendimento básico das boas práticas de arquitetura de software e das ferramentas e tecnologias envolvidas. Esse conhecimento permite interpretar e refinar as sugestões do ChatGPT, ajustando-as conforme necessário para se adequar às especificidades do projeto.

Figura 12 – Estrutura do projeto



Fonte: Adaptado pelo autor

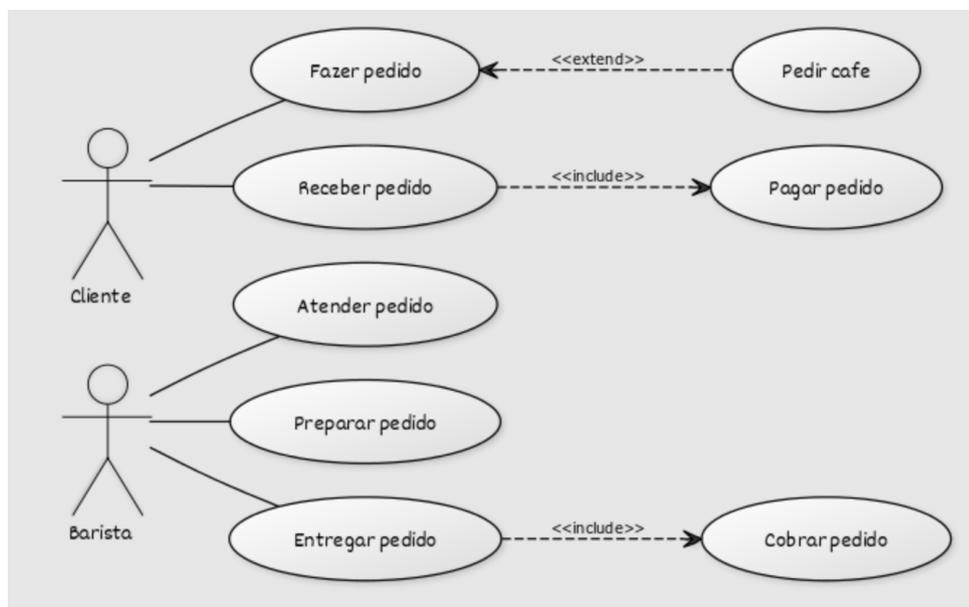
3.0.2.3 CAMADA DE DOMÍNIO

Na camada de domínio da nossa aplicação, onde se concentra a maior parte do código e das regras de negócio, o ChatGPT foi utilizado como uma ferramenta no processo de desenvolvimento de software. No entanto, no capítulo seguinte, intitulado "Análise e Resultados", apresentaremos observações detalhadas sobre como o ChatGPT influenciou positiva ou negativamente esse processo.

No diagrama de caso de uso na figura 13, podemos identificar dois atores principais: um cliente fazendo o pedido e um barista preparando o pedido. Sabendo que as portas na arquitetura hexagonal são adequadas para descrever casos de uso da aplicação, isso leva à introdução de duas portas principais: PedidoCafe e PreparandoCafe como podemos observar na figura 14. Do outro lado da aplicação, precisamos de algumas portas secundárias para armazenar os pedidos e os pagamentos. Da mesma forma, nossas portas secundárias poderiam ser chamadas simplesmente de Pedidos e Pagamentos, e sua função é armazenar e buscar pedidos e pagamentos como mostra a figura 15.

É possível observar que na classe PedidoCafe, nos preocupamos com a segurança e a confiabilidade de cada pedido. É por isso que utilizamos UUIDs (Identificadores Únicos Universais) para garantir que cada pedido seja único e rastreável.

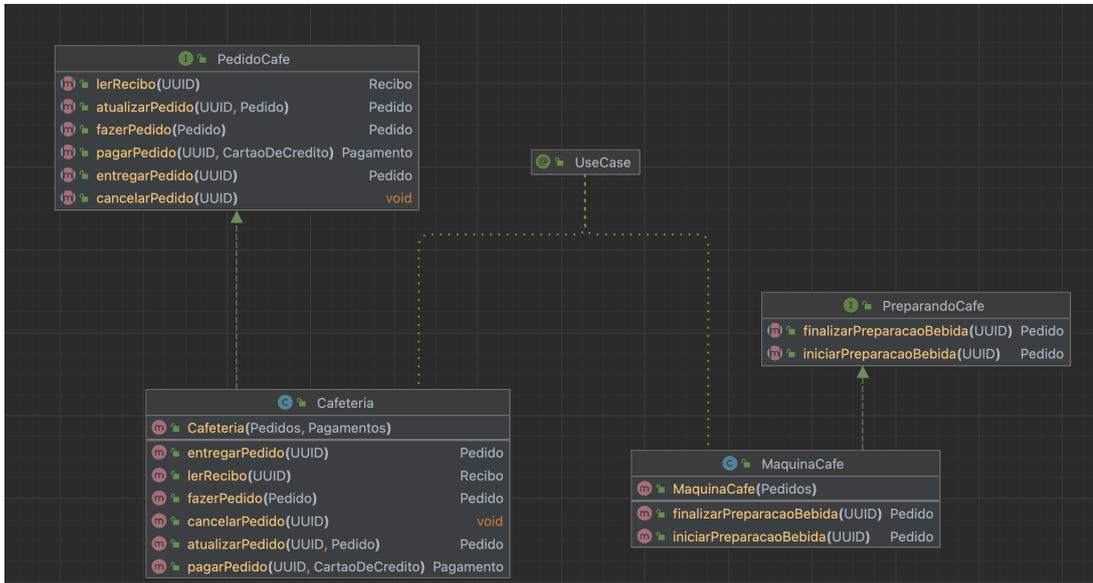
Figura 13 – Diagrama de caso de uso



Fonte: Adaptado pelo autor

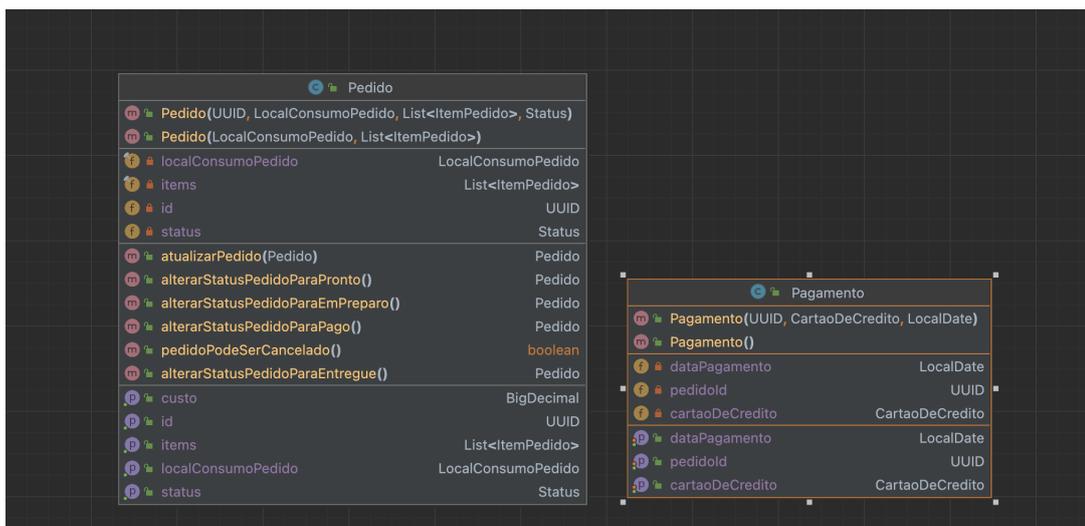
Precisaremos de algumas entidades em nosso modelo de domínio. Um item em um pedido conterá o tipo de café, leite e tamanho da bebida. Também precisaremos rastrear o status do pedido. Existem também classes para pagamentos, cartões de crédito e recibos.

Figura 14 – Diagrama de classes PedidoCafe e PreparandoCafe



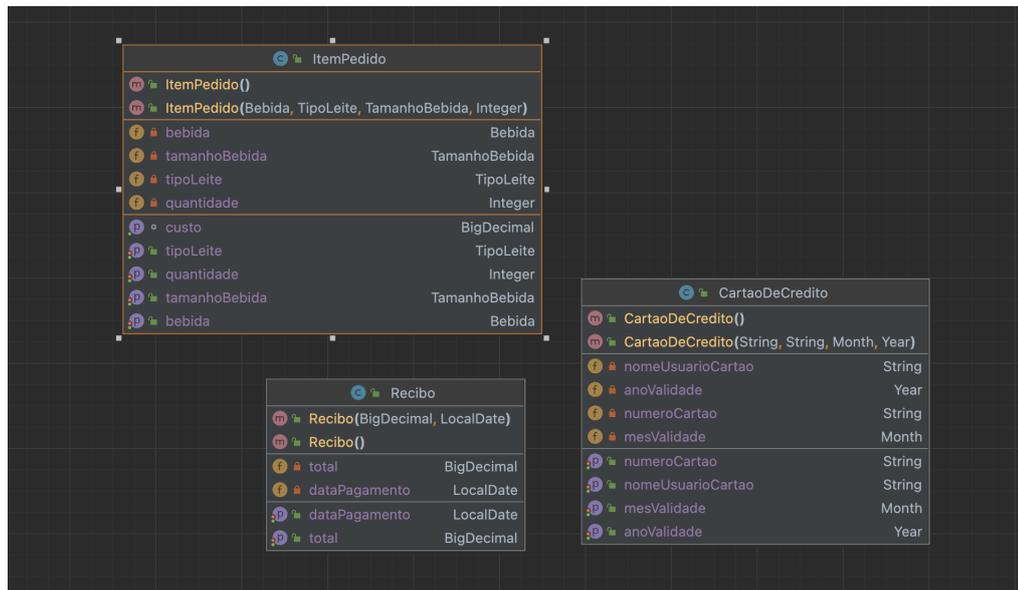
Fonte: Adaptado pelo autor

Figura 15 – Diagrama de classes Pedidos e Pagamentos



Fonte: Adaptado pelo autor

Figura 16 – Diagrama de classes ItemPedidos



Fonte: Adaptado pelo autor

Neste exemplo, não adicionamos nenhum comportamento a elas. Em seguida, precisamos implementar os casos de uso dentro da nossa aplicação. Vamos criar uma classe Cafeteria que implementa a porta primária PedidoCafe. Esta classe também chamará as portas secundárias Pedidos e Pagamentos. O papel da classe Cafeteria mostrado no Exemplo 3.1, é orquestrar operações em entidades e repositórios. Ela implementa as portas primárias como casos de uso e utiliza as portas secundárias como repositórios. Nessa implementação, a maior parte da lógica de negócios foi implementada em entidades de domínio.

```

1 @UseCase
2 public class Cafeteria implements PedidoCafe {
3
4     private final Pedidos pedidos;
5     private final Pagamentos pagamentos;
6
7     public Cafeteria(Pedidos pedidos, Pagamentos pagamentos) {
8         this.pedidos = pedidos;
9         this.pagamentos = pagamentos;
10    }
11
12    @Override
13    public Pedido fazerPedido(Pedido pedido) {
14        return pedidos.savePedido(pedido);
15    }
16
  
```

```
17     @Override
18     public Pedido atualizarPedido(UUID orderId, Pedido pedido) {
19         var pedidoExistente = pedidos.findPedidoById(orderId);
20
21         return pedidos.savePedido(pedidoExistente.atualizarPedido(pedido));
22     }
23
24     @Override
25     public void cancelarPedido(UUID pedidoId) {
26         var pedido = pedidos.findPedidoById(pedidoId);
27
28         if (!pedido.pedidoPodeSerCancelado()) {
29             throw new IllegalStateException("Pedido j  esta pago");
30         }
31
32         pedidos.deleteById(pedidoId);
33     }
34
35     @Override
36     public Pagamento pagarPedido(UUID pedidoId, CartaoDeCredito
37         cartaoDeCredito) {
38         var pedido = pedidos.findPedidoById(pedidoId);
39         pedidos.savePedido(pedido.alterarStatusPedidoParaPago());
40         return pagamentos.save(new Pagamento(pedidoId, cartaoDeCredito,
41             LocalDate.now()));
42     }
43
44     @Override
45     public Recibo lerRecibo(UUID pedidoId) {
46         var pedido = pedidos.findPedidoById(pedidoId);
47         var pagamento = pagamentos.findPagamentoByPedidoId(pedidoId);
48
49         return new Recibo(pedido.getCusto(), pagamento.dataPagamento());
50     }
51
52     @Override
53     public Pedido entregarPedido(UUID pedidoId) {
54         var pedido = pedidos.findPedidoById(pedidoId);
55
56         return pedidos.savePedido(pedido.alterarStatusPedidoParaEntregue());
57     }
58 }
```

Exemplo 3.1 – Classe Cafeteria

Como podemos observar no diagrama UML do modelo de pedido na Figura 15, toda a lógica para a mudança de status está contida no próprio modelo, ressaltando o princípio da responsabilidade única e seguindo os conceitos da arquitetura hexagonal.

Como podemos observar no Exemplo 3.2, foram usados Java Enums para essas classes, pois são constantes e seu uso é ideal para essa lógica.

```
1 public enum Bebida {
2     LATTE,
3     ESPRESSO,
4     CAPPUCCINO
5 }
6
7 public enum Status {
8     ESPERANDO_PAGAMENTO,
9     PAGO,
10    EM_PREPARO,
11    PRONTO,
12    ENTREGUE;
13 }
14
15 public enum TipoLeite {
16    INTEGRAL,
17    DESNATADO,
18    SOJA
19 }
20
21 public enum TamanhoBebida {
22    PEQUENO,
23    GRANDE
24 }
25
26 public enum LocalConsumoPedido {
27
28    VIAGEM,
29    LOJA;
30 }
```

Exemplo 3.2 – Enum criados

Para evitar o uso de um banco de dados real na classe responsável por testar, como podemos observar no seguinte Exemplo 3.5, foi utilizado um banco de dados em memória utilizando a classe *HashMap*, assim conseguimos simular o comportamento do banco de dados, o que melhora significativamente a qualidade e a eficiência dos testes.

O uso de um banco de dados em memória oferece várias vantagens. Primeiramente, ele elimina a necessidade de configurar e gerenciar um banco de dados real durante os testes, reduzindo a complexidade e o tempo necessário para a execução dos testes. Além disso, proporciona um ambiente de teste mais rápido e isolado, garantindo que os testes sejam consistentes e reproduzíveis.

```
1 public class TestDeAceitacao {
2
3     private Pedidos pedidos;
4     private Pagamentos pagamentos;
5     private PedidoCafe cafeteria;
6     private PreparandoCafe barista;
7
8     @BeforeEach
9     public void setup() {
10         pedidos = new PedidosEmMemoria();
11         pagamentos = new PagamentosEmMemoria();
12         cafeteria = new Cafeteria(pedidos, pagamentos);
13         barista = new MaquinaCafe(pedidos);
14     }
15 }
```

Exemplo 3.3 – Classe responsável por testes de aceitação

Observando os Exemplos 3.5 e 3.4, os testes são iniciados com um banco de dados em memória, o que permite uma inicialização rápida e uma execução eficiente. Esse método também facilita a criação de cenários de teste específicos, uma vez que os dados podem ser facilmente manipulados e resetados entre os testes.

```
1 public class PagamentosEmMemoria implements Pagamentos {
2
3     private final Map<UUID, Pagamento> entities = new HashMap<>();
4
5     @Override
6     public Pagamento findPagamentoByPedidoId(UUID pedidoId) {
7         return entities.get(pedidoId);
8     }
9
10    @Override
11    public Pagamento save(Pagamento pagamento) {
12        entities.put(pagamento.pedidoId(), pagamento);
13        return pagamento;
14    }
15 }
```

```
14 }
15 }
```

Exemplo 3.4 – Pagamentos em memória

```
1 public class PedidosEmMemoria implements Pedidos {
2
3     private final Map<UUID, Pedido> entities = new HashMap<>();
4
5     @Override
6     public Pedido findPedidoById(UUID pedidoId) throws PedidoNaoEncontrado
7     {
8         var order = entities.get(pedidoId);
9
10        if (Objects.isNull(order))
11            throw new PedidoNaoEncontrado();
12
13        return order;
14    }
15
16    @Override
17    public Pedido savePedido(Pedido pedido) {
18        entities.put(pedido.getId(), pedido);
19        return pedido;
20    }
21
22    @Override
23    public void deleteById(UUID pedidoId) {
24        entities.remove(pedidoId);
25    }
26 }
```

Exemplo 3.5 – Pedidos em memória

Seguindo os requisitos mencionados anteriormente, foram criados testes unitários para garantir a qualidade do código e verificar a correta implementação das regras de negócios conforme definidas. Esses testes unitários são essenciais para assegurar que cada componente do sistema funcione isoladamente como esperado, facilitando a identificação de problemas específicos.

Além disso, a adoção de testes unitários permite uma manutenção mais eficiente do código, pois eventuais mudanças podem ser verificadas automaticamente para garantir que não introduzam novos erros. Isso é especialmente importante em projetos que seguem

metodologias ágeis, como o Scrum, onde as entregas contínuas e incrementais são uma prática comum.

O uso de ferramentas de automação de testes, como JUnit para testes unitários foi fundamental para agilizar o processo de verificação e validação do sistema. Esses testes automatizados são executados regularmente em pipelines de integração contínua, garantindo um ciclo de desenvolvimento mais robusto e confiável.

```
1 @Test
2 void cliente_pode_fazer_pedido() {
3     var pedidoFeito = new Pedido(LocalConsumoPedido.LOJA,
4         List.of(new ItemPedido(Bebida.CAPPUCCINO, TipoLeite.DESNATADO,
5             TamanhoBebida.PEQUENO, 1)));
6
7     var compra = cafeteria.fazerPedido(pedidoFeito);
8
9     Assertions.assertThat(compra.getLocalConsumoPedido())
10        .isEqualTo(LocalConsumoPedido.LOJA);
11
12    assertThat(compra.getItems())
13        .containsExactly(new ItemPedido(Bebida.CAPPUCCINO, TipoLeite.
14            DESNATADO, TamanhoBebida.PEQUENO, 1));
15
16    assertThat(compra.getStatus()).isEqualTo(Status.ESPERANDO_PAGAMENTO);
17 }
18
19 @Test
20 void cliente_pode_atualizar_pedido_antes_pagamento() {
21     var compraComUmItem = new Pedido(LocalConsumoPedido.VIAGEM,
22         List.of(new ItemPedido(Bebida.LATTE, TipoLeite.INTEGRAL,
23             TamanhoBebida.GRANDE, 1)));
24
25     var compraComDoisItems = new Pedido(LocalConsumoPedido.VIAGEM,
26         List.of(new ItemPedido(Bebida.LATTE, TipoLeite.INTEGRAL,
27             TamanhoBebida.GRANDE, 2)));
28
29     var compra = cafeteria.fazerPedido(compraComUmItem);
30     var compraAtualizada = cafeteria.atualizarPedido(compra.getId(),
31         compraComDoisItems);
32
33     assertThat(compraAtualizada.getItems())
34        .containsExactly(new ItemPedido(Bebida.LATTE, TipoLeite.
35            INTEGRAL, TamanhoBebida.GRANDE, 2));
36 }
```

```
33 @Test
34 void cliente_pode_cancelar_pedido_antes_do_pagamento() {
35     var pedidoSalvo = pedidos.savePedido(umPedido());
36
37     cafeteria.cancelarPedido(pedidoSalvo.getId());
38
39     assertThatThrownBy(() -> pedidos.findPedidoById(pedidoSalvo.getId()))
40         .isInstanceOf(PedidoNaoEncontrado.class);
41 }
42 @Test
43 void cliente_pode_pagar_pedido() {
44     var pedidoSalvo = pedidos.savePedido(umPedido());
45     var cartaoDeCredito = CartaoDeCreditoTestFactory.umCartaoDeCredito();
46
47     var pagamento = cafeteria.pagarPedido(pedidoSalvo.getId(),
48         cartaoDeCredito);
49
50     assertThat(pagamento.pedidoId()).isEqualTo(pedidoSalvo.getId());
51     assertThat(pagamento.cartaoDeCredito()).isEqualTo(cartaoDeCredito);
52     assertThat(pedidos.findPedidoById(pedidoSalvo.getId()).getStatus())
53         .isEqualTo(Status.PAGO);
54 }
55 @Test
56 void nenhuma_mudanca_e_possivel_apos_pagamento() {
57     var pedidoSalvo = pedidos.savePedido(umPedidoPago());
58
59     assertThatThrownBy(() -> cafeteria.atualizarPedido(pedidoSalvo.getId(),
60         umPedido())).isInstanceOf(IllegalStateException.class);
61 }
62 @Test
63 void cliente_pode gerar_recibo_quando_a_pedido_esta_pago() {
64     var pedidoSalvo = pedidos.savePedido(umPedidoPago());
65     var pagamentoSalvo = pagamentos.save(PagamentosTestFactory.
66         umPagamentoParaPedido(pedidoSalvo));
67
68     var receipt = cafeteria.lerRecibo(pedidoSalvo.getId());
69
70     assertThat(receipt.total()).isEqualTo(pedidoSalvo.getCusto());
71     assertThat(receipt.dataPagamento()).isEqualTo(pagamentoSalvo.
72         dataPagamento());
73 }
74 @Test
75 void barista_pode_preparar_bebida_apos_pedido_ser_pago() {
```

```
74     var pedidoSalvo = pedidos.savePedido(umPedidoPago());
75
76     var orderInPreparation = barista.iniciarPreparacaoBebida(pedidoSalvo.
77         getId());
78     assertThat(orderInPreparation.getStatus()).isEqualTo(Status.EM_PREPARO)
79         ;
80 }
81 @Test
82 void
83     barista_pode_alterar_status_do_pedido_para_pronto_quando_finalizar_preparacao() {
84     var pedidoSalvo = pedidos.savePedido(umPedidoEmPreparacao());
85
86     var preparedOrder = barista.finalizarPreparacaoBebida(pedidoSalvo.getId());
87
88     assertThat(preparedOrder.getStatus()).isEqualTo(Status.PRONTO);
89 }
90 @Test
91 void cliente_pode_retirar_pedido_apos_status_pronto() {
92     var pedidoSalvo = pedidos.savePedido(umPedidoPronto());
93
94     var takenOrder = cafeteria.entregarPedido(pedidoSalvo.getId());
95
96     assertThat(takenOrder.getStatus()).isEqualTo(Status.ENTREGUE);
97 }
```

Exemplo 3.6 – Teste unitário - Principais Casos de Uso

Observa-se no Exemplo 3.6 que foram testados os principais cenários, como quando um cliente tenta adicionar um item a um pedido já pago. Isso destaca uma grande vantagem de utilizar essa arquitetura: a capacidade de focar principalmente nas regras de negócio na camada de domínio, abstraindo assim qualquer problema relacionado ao desenvolvimento e teste de frameworks e bibliotecas. Além disso, essa abordagem oferece vários outros benefícios:

- **Isolamento de Complexidade:** Ao manter as regras de negócio separadas da infraestrutura, facilitamos a manutenção e a evolução do sistema, pois mudanças em bibliotecas ou frameworks não impactam diretamente o núcleo da aplicação.
- **Testabilidade:** Com as regras de negócio isoladas, é mais fácil criar testes unitários e

de integração, garantindo que a lógica essencial do negócio funcione corretamente independentemente das mudanças tecnológicas.

- **Reutilização de Código:** A camada de domínio pode ser reutilizada em diferentes contextos e aplicações, promovendo a consistência e reduzindo o esforço de desenvolvimento.
- **Flexibilidade Tecnológica:** A separação das camadas permite trocar componentes de infraestrutura sem necessidade de grandes refatorações na lógica de negócio. Por exemplo, podemos mudar o banco de dados ou o framework web sem alterar a camada de domínio.
- **Evolução Gradual:** Permite a implementação de melhorias e novas funcionalidades de forma incremental, minimizando riscos e permitindo que o sistema se adapte rapidamente às novas necessidades do negócio.
- **Facilidade de Colaboração:** A clara divisão entre as camadas facilita o trabalho em equipe, permitindo que desenvolvedores foquem em suas áreas de especialização, seja na lógica de negócio ou na infraestrutura.

Adotar uma arquitetura que prioriza a camada de domínio resulta em um sistema mais robusto, adaptável e fácil de manter, permitindo que as empresas respondam rapidamente às mudanças do mercado e melhorem continuamente seus processos e produtos.

3.0.2.4 CAMADA DE INFRAESTRUTURA

Seguimos o mesmo padrão na camada de infraestrutura, utilizando as portas PedidoCafe e PreparandoCafe. Desta vez, as portas são usadas em controladores que recebem requisições HTTP e empregam adaptadores para salvar dados em bancos de dados. Em nossa aplicação, estamos utilizando um banco de dados em memória, conhecido como H2 e de fácil utilização, eliminando a necessidade de criar scripts SQL para gerar tabelas ou inserir dados. Além disso, estamos utilizando o Spring Boot para gerenciar todo o processo de tratamento e processamento de requisições HTTP e conexões com o banco de dados. Isso torna a aplicação bastante prática e permite seu desenvolvimento em menor tempo, em comparação com aplicações que não utilizam frameworks para essas tarefas.

- **Escalabilidade facilitada:** Com o Spring Boot, a escalabilidade é simplificada, pois ele oferece ferramentas e configurações prontas para otimizar o desempenho da aplicação conforme a demanda aumenta.
- **Padronização:** Utilizar um framework como o Spring Boot proporciona uma padronização do código, o que facilita a manutenção e a colaboração entre desenvolvedores, além de reduzir a curva de aprendizado para novos integrantes da equipe.

- **Segurança:** O Spring Boot inclui vários mecanismos de segurança integrados, como autenticação e autorização, que podem ser facilmente configurados para proteger a aplicação.
- **Automação e configuração simplificadas:** Com o Spring Boot, muitas configurações são automatizadas, o que reduz a necessidade de configuração manual e permite que os desenvolvedores foquem mais nas funcionalidades de negócio.
- **Testes simplificados:** O Spring Boot facilita a criação de testes automatizados, oferecendo suporte para testes unitários, de integração e end-to-end, garantindo que a aplicação funcione corretamente em todas as etapas de desenvolvimento.
- **Redução de custos:** A utilização de ferramentas e frameworks que automatizam processos e reduzem a complexidade do desenvolvimento pode resultar em uma significativa redução de custos, tanto em termos de tempo quanto de recursos humanos.

```
1 @RestController
2 @RequestMapping("/api/v1/pedido")
3 @RequiredArgsConstructor
4 public class PedidoController {
5
6     private final PedidoCafe pedidoCafe;
7
8     @PostMapping
9     public ResponseEntity<OrderResponse> criarPedido(@RequestBody
10         PedidoRequisicao request, UriComponentsBuilder uriComponentsBuilder)
11     {
12         var order = pedidoCafe.fazerPedido(request.toDomain());
13         var location = uriComponentsBuilder.path("/pedido/{id}")
14             .buildAndExpand(order.getId())
15             .toUri();
16         return ResponseEntity.created(location).body(OrderResponse.
17             fromDomain(order));
18     }
19
20     @PutMapping("/{id}")
21     public ResponseEntity<OrderResponse> atualizarPedido(@PathVariable UUID
22         id, @RequestBody PedidoRequisicao request) {
23         var order = pedidoCafe.atualizarPedido(id, request.toDomain());
24         return ResponseEntity.ok(OrderResponse.fromDomain(order));
25     }
26
27     @DeleteMapping("/{id}")
28     public ResponseEntity<Void> cancelarPedido(@PathVariable UUID id) {
```

```
25     pedidoCafe.cancelarPedido(id);
26     return ResponseEntity.noContent().build();
27 }
28 }
```

Exemplo 3.7 – Controlador do pedido

No Exemplo 3.7, é possível observar a implementação de funções para criar, deletar e atualizar pedidos, todas utilizando os verbos HTTP apropriados para cada operação. A função de criação de pedido usa o verbo POST, a de deleção usa DELETE, e a de atualização utiliza PUT. Esses verbos foram escolhidos de acordo com as melhores práticas de *RESTful APIs*. Adicionalmente, o endpoint "api/v1/pedido" está sendo utilizado, o que demonstra a adoção de boas práticas de versionamento de API. O versionamento é crucial para manter a compatibilidade e facilitar a evolução da API sem quebrar funcionalidades existentes para os usuários.

```
1 @Component
2 @RequiredArgsConstructor
3 public class PedidosJpaAdapter implements Pedidos {
4
5     private final PedidosJpaRepository pedidosJpaRepository;
6
7     @Override
8     public Pedido findPedidoById(UUID pedidoId) throws PedidoNaoEncontrado
9     {
10         return pedidosJpaRepository.findById(pedidoId)
11             .map(PedidoEntidade::toDomain)
12             .orElseThrow(PedidoNaoEncontrado::new);
13     }
14
15     @Override
16     public Pedido savePedido(Pedido pedido) {
17         return pedidosJpaRepository.save(PedidoEntidade.fromDomain(pedido)
18             ).toDomain();
19     }
20
21     @Override
22     public void deleteById(UUID pedidoId) {
23         pedidosJpaRepository.deleteById(pedidoId);
24     }
25 }
```

Exemplo 3.8 – Adaptador do pedido

No Exemplo 3.8, é possível observar que, no adaptador criado para a função de busca de pedidos pelo número de identificação único UUID, está sendo utilizada a classe `Optional` do Java. Esse uso evidencia que versões recentes do Java estão sendo empregadas, o que traz várias vantagens. A classe `Optional` melhora significativamente o manuseio de valores nulos, permitindo que os desenvolvedores evitem problemas comuns relacionados a `null`, como o temido `NullPointerException`.

Além disso, o uso de `Optional` facilita a implementação de tratamentos de exceções mais elegantes e robustos. Ao encapsular valores que podem ser ou não presentes em um `Optional`, pode-se utilizar métodos como `isPresent` e `ifPresent`, que promovem uma abordagem mais fluente e segura ao lidar com possíveis ausências de dados.

Outro ponto importante é que o uso de `Optional` se alinha com as boas práticas de programação funcional que estão cada vez mais sendo incorporadas nas linguagens modernas. Com isso, os desenvolvedores têm à disposição ferramentas para criar código mais expressivo e conciso.

```
1 @RestResourceTest
2 public class PedidoControllerTest {
3
4     @Autowired
5     private MockMvc mockMvc;
6
7     @Autowired
8     private Pedidos pedidos;
9
10    private final String orderJson = ""
11        {
12            "localConsumoPedido": "LOJA",
13            "items": [
14                {
15                    "bebida": "LATTE",
16                    "quantidade": 1,
17                    "tipoLeite": "INTEGRAL",
18                    "tamanhoBebida": "GRANDE"
19                }
20            ]
21        }
22    """;
23
24    @Test
25    void createOrder() throws Exception {
26        mockMvc.perform(post("/api/v1/pedido")
27            .contentType(MediaType.APPLICATION_JSON_VALUE)
28            .content(orderJson))
```

```
29         .andExpect(status().isCreated());
30     }
31
32     @Test
33     void updateOrder() throws Exception {
34         var order = pedidos.savePedido(umPedido());
35
36         mockMvc.perform(put("/api/v1/pedido/{id}", order.getId())
37             .contentType(MediaType.APPLICATION_JSON_VALUE)
38             .content(orderJson))
39             .andExpect(status().isOk());
40     }
41
42     @Test
43     void cancelOrder() throws Exception {
44         var order = pedidos.savePedido(umPedido());
45
46         mockMvc.perform(delete("/api/v1/pedido/{id}", order.getId()))
47             .andExpect(status().isNoContent());
48     }
49 }
```

Exemplo 3.9 – Teste de integração do controlador de pedido

Como demonstrado no Exemplo 3.9, observa-se que testes de integração foram realizados aproveitando as vantagens proporcionadas pelo Spring Boot. Esse framework oferece um conjunto robusto de ferramentas para a execução de testes de integração, permitindo que simulemos requisições e verifiquemos as respostas recebidas em diferentes cenários.

A utilização do Spring Boot nesse contexto é extremamente benéfica, pois ele simplifica a configuração e a execução de testes complexos, garantindo que todas as camadas da aplicação sejam testadas de forma integrada. Com o Spring Boot, pode-se configurar facilmente um ambiente de teste que replica as condições de produção, o que é essencial para validar o comportamento do sistema em situações reais.

Além disso, o Spring Boot permite a criação de testes que interagem com a base de dados, os serviços e os componentes da aplicação, assegurando que todas as partes do sistema funcionem harmoniosamente. Isso inclui a simulação de diferentes tipos de requisições HTTP e a verificação das respostas esperadas para cada cenário específico. Tal abordagem não apenas valida a funcionalidade isolada dos componentes, mas também a sua interação correta, proporcionando maior confiança na estabilidade e na qualidade do sistema como um todo.

Com a metodologia exposta ao longo do capítulo e o uso do ChatGPT descrito em cada etapa do desenvolvimento de software, chegamos agora ao ponto crucial de nosso estudo. A análise dos dados coletados e os resultados obtidos são fundamentais para entender o real impacto da ferramenta no processo de desenvolvimento.

4 Apresentação e Análise de Resultados

Este capítulo apresenta a essência do nosso estudo sobre "ChatGPT: Uma análise da ferramenta aplicada no processo de desenvolvimento de software". Aqui, nos concentraremos em análises realizadas para compreender o impacto e a eficácia do ChatGPT em diversos aspectos do desenvolvimento de software.

Durante o curso desta pesquisa, exploram-se as nuances do uso do ChatGPT em várias fases do processo de desenvolvimento, desde a concepção inicial até a implementação e manutenção contínua. Nossa análise abrangeu aspectos como a qualidade do código gerado, a eficiência na resolução de problemas

Ao longo deste capítulo, serão detalhadas as metodologias empregadas para coletar dados, os parâmetros de avaliação estabelecidos e os resultados obtidos a partir dessas análises. Através de uma abordagem ampla, buscamos oferecer insights valiosos sobre como o ChatGPT pode ser uma ferramenta transformadora no contexto do desenvolvimento de software.

4.0.1 CHATGPT NA COLETA DE REQUISITOS

Figura 17 – User story - requisito 1

Crie uma story scrum para o requisito de uma aplicacao para cafeteria. "O cliente pode pedir um café e escolher o tipo de café, leite, tamanho e se é para consumir no local ou para levar"



História do Usuário:

Como cliente, eu quero **pedir um café** em uma cafeteria para que eu possa **personalizá-lo de acordo com minhas preferências e decidir se quero consumir no local ou levar**.

Critérios de Aceitação:

1. Como cliente, desejo selecionar o tipo de café que desejo (expresso, cappuccino, latte, etc.).
2. Como cliente, desejo escolher o tipo de leite para o meu café (integral, desnatado, soja, etc.).
3. Como cliente, desejo selecionar o tamanho do meu café (pequeno, médio, grande).
4. Como cliente, desejo ter a opção de consumir meu café no local ou para levar.
5. Como cliente, desejo ter a opção de adicionar extras ao meu café, como xaropes ou chantilly (se disponíveis).
6. Como cliente, desejo ter a opção de ver um resumo do meu pedido antes de finalizá-lo.
7. Como cliente, desejo ter a opção de modificar meu pedido antes de finalizá-lo.
8. Como cliente, desejo ter a opção de cancelar meu pedido antes de finalizá-lo.

Fonte: Adaptado pelo autor

Iniciando pela coleta de requisitos, foi solicitado ao ChatGPT criar uma user story do seguinte requisito 17. Como se pode observar, sem fornecer muitos detalhes sobre o requisito, o ChatGPT ainda consegue nos trazer uma user story bem descrita e compreensível para ser transformada em uma tarefa para um desenvolvedor.

No entanto, se o profissional que estiver utilizando o ChatGPT para colher requisitos não tiver um bom entendimento do produto ou dos próprios requisitos, o ChatGPT pode apresentar informações que talvez não façam parte dos requisitos reais, trazendo assim ambiguidades ou até mesmo funcionalidades desnecessárias para a aplicação.

Essa limitação pode resultar em especificações confusas ou mal direcionadas, que complicam o processo de desenvolvimento em vez de simplificá-lo. A falta de clareza sobre o escopo do projeto e os objetivos específicos pode levar a uma proliferação de funcionalidades que não agregam valor real ao produto final, desviando o foco dos desenvolvedores das necessidades essenciais dos usuários.

Além disso, o risco de ambiguidades aumenta quando o profissional não possui uma compreensão sólida dos conceitos técnicos e funcionais do projeto. O ChatGPT, embora poderoso, baseia suas respostas em padrões e exemplos encontrados em sua base de conhecimento, o que pode incluir informações irrelevantes ou descontextualizadas se não for guiado corretamente. Essa situação requer que o profissional utilize uma combinação de habilidades de interpretação e validação para filtrar e ajustar as sugestões geradas pela IA.

Para mitigar esses riscos, é crucial que o profissional encarregado da coleta de requisitos tenha uma compreensão clara e profunda do produto, dos objetivos do projeto e das necessidades dos usuários finais. Ele deve ser capaz de formular perguntas precisas e avaliar criticamente as respostas fornecidas pelo ChatGPT, garantindo que apenas informações pertinentes sejam incorporadas aos requisitos do projeto.

Além disso, é recomendável que o processo de coleta de requisitos com o ChatGPT seja complementado por métodos tradicionais de elicitación, como entrevistas, workshops e análise de documentos, para validar e enriquecer as informações obtidas. A integração dessas abordagens pode ajudar a assegurar que os requisitos capturados sejam completos, precisos e alinhados com as expectativas do cliente e dos stakeholders.

4.0.2 CHATGPT NO PROCESSO DE DESENVOLVIMENTO DE CÓDIGO E TESTES

Durante o processo de desenvolvimento de código, o ChatGPT foi utilizado de forma abrangente, desde a modelagem de software até o desenvolvimento de testes unitários, de integração e criação de métodos e classes. Essa abordagem permitiu explorar o potencial da inteligência artificial em diversas etapas do ciclo de vida do software, demonstrando tanto suas capacidades quanto suas limitações.

Figura 18 – Prompt para criar classe Pedido utilizando ChatGPT

Criar uma classe utilizando a linguagem Java, que tenha metodos get e set, tenha as seguintes propriedades, id: UUID, items: List<ItemPedido>, status: Status

 Para criar uma classe em Java que tenha as propriedades `id` (do tipo `UUID`), `items` (do tipo `List<ItemPedido>`) e `status` (do tipo `Status`), precisamos definir a classe com seus atributos, construtores, métodos `get` e `set`, e possivelmente uma enumeração `Status` e uma classe `ItemPedido` para complementar.

A seguir, apresento um exemplo completo dessa estrutura:

1. Classe `ItemPedido`
2. Enumeração `Status`
3. Classe `Pedido` com os atributos solicitados e métodos `get` e `set`

Fonte: Adaptado pelo autor

```
1 import java.util.List;
2 import java.util.UUID;
3
4 public class Pedido {
5     private UUID id;
6     private List<ItemPedido> items;
7     private Status status;
8
9     // Construtor
10    public Pedido(UUID id, List<ItemPedido> items, Status status) {
11        this.id = id;
12        this.items = items;
13        this.status = status;
14    }
15
16    // Getters e Setters
17    public UUID getId() {
18        return id;
19    }
20
21    public void setId(UUID id) {
22        this.id = id;
23    }
24
25    public List<ItemPedido> getItems() {
26        return items;
27    }
```

```
28
29     public void setItems(List<ItemPedido> items) {
30         this.items = items;
31     }
32
33     public Status getStatus() {
34         return status;
35     }
36
37     public void setStatus(Status status) {
38         this.status = status;
39     }
40 }
```

Exemplo 4.1 – Classe Pedido gerado pelo ChatGPT

```
1 public enum Status {
2     PENDENTE ,
3     PROCESSANDO ,
4     ENVIADO ,
5     ENTREGUE ,
6     CANCELADO
7 }
```

Exemplo 4.2 – Java Enum Status gerado pelo ChatGPT

Como se pode notar no Exemplo 18, ao solicitar que o ChatGPT criasse uma classe, foi necessário especificar os tipos de dados das propriedades das classes. No entanto, o ChatGPT foi capaz de gerar a enumeração Status sem precisar atribuir explicitamente valores ou tipos para suas propriedades, demonstrando sua capacidade de abstração e entendimento das convenções comuns em programação. Além disso, pode-se observar que os métodos construtores, bem como os métodos get e set, foram gerados corretamente, respeitando as práticas recomendadas de encapsulamento e acessibilidade em Java.

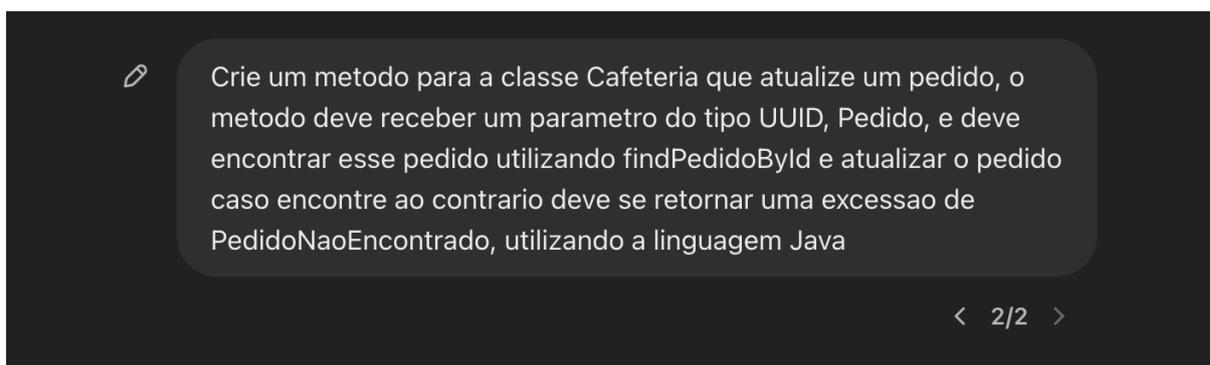
A classe principal, Pedido, integra as propriedades id (um identificador único do tipo UUID), items (uma lista de objetos do tipo ItemPedido) e status (do tipo Status). O construtor da classe Pedido inicializa essas propriedades, e os métodos get e set permitem o acesso e a modificação dessas propriedades de maneira segura e controlada. Essa classe encapsula todas as informações relevantes de um pedido, proporcionando uma estrutura clara e organizada para a manipulação de dados de pedidos em um sistema.

Em suma, a estrutura gerada pelo ChatGPT demonstra uma compreensão sólida dos princípios de orientação a objetos, como encapsulamento, abstração e definição de tipos.

A implementação segue as boas práticas de desenvolvimento em Java, garantindo que o código seja claro, conciso e fácil de manter.

É importante observar que o usuário que tenta criar uma classe utilizando o GPT deve possuir um conhecimento básico em desenvolvimento de software, especialmente em programação orientada a objetos e na linguagem Java. A criação de classes, a definição de atributos e métodos, e o entendimento de conceitos como encapsulamento e herança são fundamentais para aproveitar ao máximo as capacidades do GPT.

Figura 19 – Prompt para criar método para atualizar pedido



Fonte: Adaptado pelo autor

Durante o processo de desenvolvimento, especialmente ao lidar com tarefas mais específicas como na Figura 19, como a criação de funções para regras de negócio particulares, percebemos a necessidade de fornecer detalhes mais precisos ao ChatGPT. Por exemplo, neste projeto, ajustes tiveram que ser feitos em todos os códigos gerados pelo ChatGPT para garantir que atendessem às exigências específicas. No entanto, notamos que quando se trata de funções menos específicas, como determinar se uma String contém um determinado caractere, o ChatGPT pode oferecer resultados mais concisos e eficazes.

Ao analisar o Exemplo 4.3, também observamos que o ChatGPT não realiza a separação de arquivos por classe. Portanto, é responsabilidade do profissional que utiliza o ChatGPT organizar e estruturar o código conforme as convenções e práticas adequadas de programação. Essa tarefa inclui a separação dos arquivos de código-fonte em classes distintas, garantindo assim uma organização e manutenção mais eficientes do projeto.

```
1 import java.util.HashMap;
2 import java.util.Map;
3 import java.util.UUID;
4
5 class Pedido {
6     // Atributos e m todos da classe Pedido
7 }
```

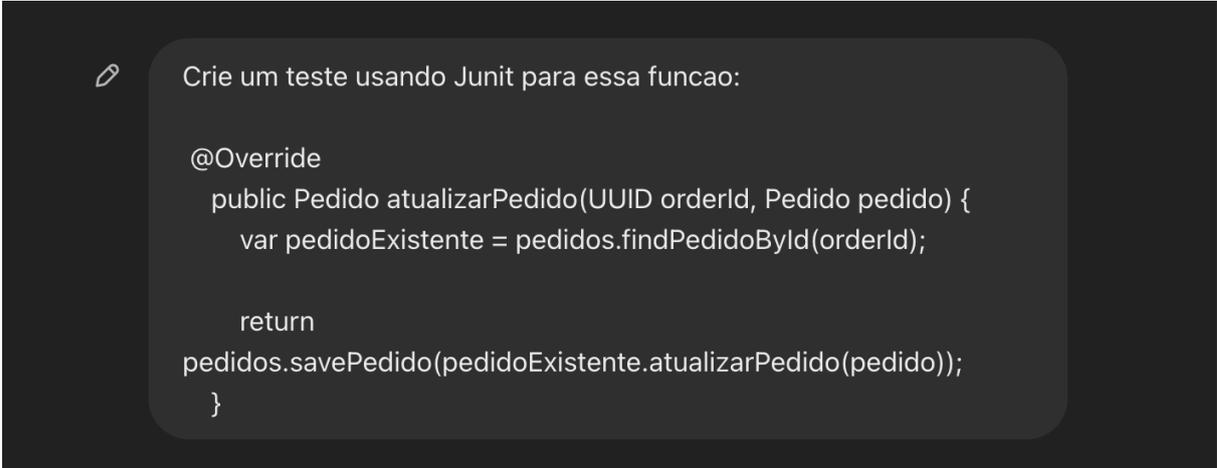
```
8
9 class PedidoNaoEncontradoException extends Exception {
10     public PedidoNaoEncontradoException(String mensagem) {
11         super(mensagem);
12     }
13 }
14
15 class Cafeteria {
16     private Map<UUID, Pedido> pedidos;
17
18     public Cafeteria() {
19         this.pedidos = new HashMap<>();
20     }
21
22     // M todo para adicionar um pedido
23     public void adicionarPedido(UUID id, Pedido pedido) {
24         this.pedidos.put(id, pedido);
25     }
26
27     // M todo para encontrar um pedido por ID
28     public Pedido findPedidoById(UUID id) throws
29         PedidoNaoEncontradoException {
30         Pedido pedido = this.pedidos.get(id);
31         if (pedido == null) {
32             throw new PedidoNaoEncontradoException("Pedido n o encontrado
33                 com o ID: " + id);
34         }
35         return pedido;
36     }
37
38     // M todo para atualizar um pedido
39     public void atualizarPedido(UUID id, Pedido novoPedido) throws
40         PedidoNaoEncontradoException {
41         if (this.pedidos.containsKey(id)) {
42             this.pedidos.put(id, novoPedido);
43         } else {
44             throw new PedidoNaoEncontradoException("Pedido n o encontrado
45                 com o ID: " + id);
46         }
47     }
48 }
```

Exemplo 4.3 – Código gerado após o prompt

Ao utilizar o ChatGPT para escrever testes unitários e de integração, exemplo da

Figura 20, os desenvolvedores podem se beneficiar da capacidade do modelo de gerar casos de teste com base nos prompts fornecidos. Esse processo pode ser especialmente útil para agilizar o desenvolvimento de testes em situações onde é necessário cobrir diferentes cenários de uso.

Figura 20 – Prompt para criar teste de caso de uso



```
Crie um teste usando Junit para essa funcao:

@Override
public Pedido atualizarPedido(UUID orderId, Pedido pedido) {
    var pedidoExistente = pedidos.findPedidoById(orderId);

    return
pedidos.savePedido(pedidoExistente.atualizarPedido(pedido));
}
```

Fonte: Adaptado pelo autor

No entanto, é importante ressaltar que, embora o ChatGPT possa gerar o esqueleto do teste como no Exemplo 4.4, ainda é necessário que os profissionais tenham um entendimento sólido das ferramentas de teste disponíveis na linguagem de programação utilizada. Isso inclui conhecimento sobre bibliotecas como JUnit para testes em Java, ou frameworks como pytest para Python, entre outros.

```
1 import org.junit.jupiter.api.Test;
2 import static org.junit.jupiter.api.Assertions.*;
3 import static org.mockito.Mockito.*;
4
5 class PedidoServiceTest {
6
7     @Test
8     void testAtualizarPedido() {
9         // Criar instancia do servi o de pedidos
10        PedidoRepository mockPedidoRepository = mock(PedidoRepository.class
11        );
12        PedidoService pedidoService = new PedidoService(
13        mockPedidoRepository);
14
15        // Criar um pedido de exemplo e um UUID
16        UUID orderId = UUID.randomUUID();
```

```
15     Pedido pedidoAtualizado = new Pedido(/* Informa es do pedido
16         atualizado */);
17
18     // Mock do pedido existente retornado pelo reposit rio
19     Pedido pedidoExistente = new Pedido(/* Informa es do pedido
20         existente */);
21     when(mockPedidoRepository.findById(orderId)).thenReturn(
22         pedidoExistente);
23
24     // Chamar o m todo atualizarPedido
25     Pedido pedidoRetornado = pedidoService.atualizarPedido(orderId,
26         pedidoAtualizado);
27
28     // Verificar se o m todo savePedido foi chamado com o pedido
29     // correto
30     verify(mockPedidoRepository).savePedido(eq(pedidoExistente.
31         atualizarPedido(pedidoAtualizado)));
32
33     // Verificar se o pedido retornado o mesmo que foi salvo
34     assertEquals(pedidoExistente.atualizarPedido(pedidoAtualizado),
35         pedidoRetornado);
36 }
37 }
```

Exemplo 4.4 – Código gerado após o prompt para gerar teste

Além disso, compreender as regras de negócio subjacentes ao código é essencial para garantir que os testes cubram adequadamente todos os casos de uso relevantes. Os testes devem verificar não apenas a corretude técnica do código, mas também se ele está atendendo aos requisitos funcionais e de negócio estabelecidos.

Um aspecto crítico é o entendimento da arquitetura do software em questão. Isso inclui compreender como os diferentes componentes se comunicam e interagem entre si. Em muitos casos, pode ser necessário simular o comportamento de componentes externos, como APIs de terceiros, bancos de dados ou sistemas legados, para garantir que os testes sejam confiáveis e independentes.

4.1 RESULTADOS

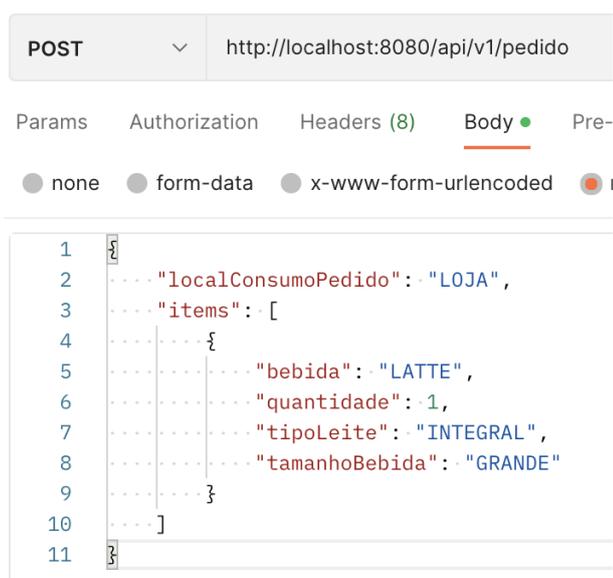
Conforme discutido anteriormente em diversos pontos deste trabalho, é possível empregar o GPT como uma ferramenta no ciclo de desenvolvimento de software. No entanto, é crucial compreender que o GPT não deve ser utilizado como a única ferramenta em qualquer

etapa do processo, seja na elicitação de requisitos, no desenvolvimento de código ou nos testes.

O uso eficaz do ChatGPT requer que seja empregado por um profissional capacitado e que possua conhecimento específico na tarefa em questão. Embora o ChatGPT possa melhorar o desempenho do profissional de desenvolvimento de software, ao gerar o esqueleto do código e economizar tempo em tarefas mais genéricas, como a definição de funções ou regras de negócio, ele não substitui o conhecimento humano e a experiência técnica.

É importante reconhecer que o ChatGPT é uma ferramenta poderosa quando utilizada de forma complementar e inteligente. Seu papel pode ser de auxílio na geração de ideias, na exploração de soluções criativas e na aceleração de determinadas tarefas. No entanto, para garantir a qualidade e a precisão do trabalho final, é fundamental que o profissional exerça o controle sobre o processo e aplique sua expertise para validar e refinar as saídas geradas pelo modelo.

Figura 21 – Requisição HTTP para criar um novo pedido



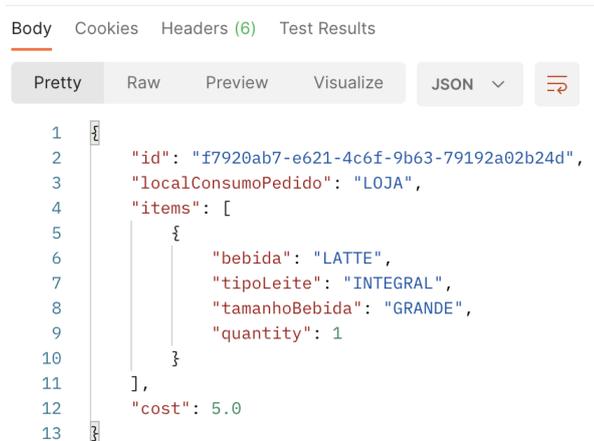
Fonte: Adaptado pelo autor

Além disso, realizamos testes no projeto por meio de requisições HTTP utilizando o Postman como ferramenta. Na Figura 21, é evidente que um pedido foi criado com êxito, pois temos uma resposta HTTP status 201 na Figura 22, o que indica criado com sucesso. Além disso, pode-se observar que todos os testes foram executados sem falhas, garantindo a integridade do projeto. Para acessar o repositório do projeto, ele está disponível no seguinte [link](#).

Na Figura 23, é evidente que o projeto foi compilado com sucesso, o que indica a integridade do código-fonte.

Este projeto inicialmente concebido para testar as capacidades do ChatGPT acabou

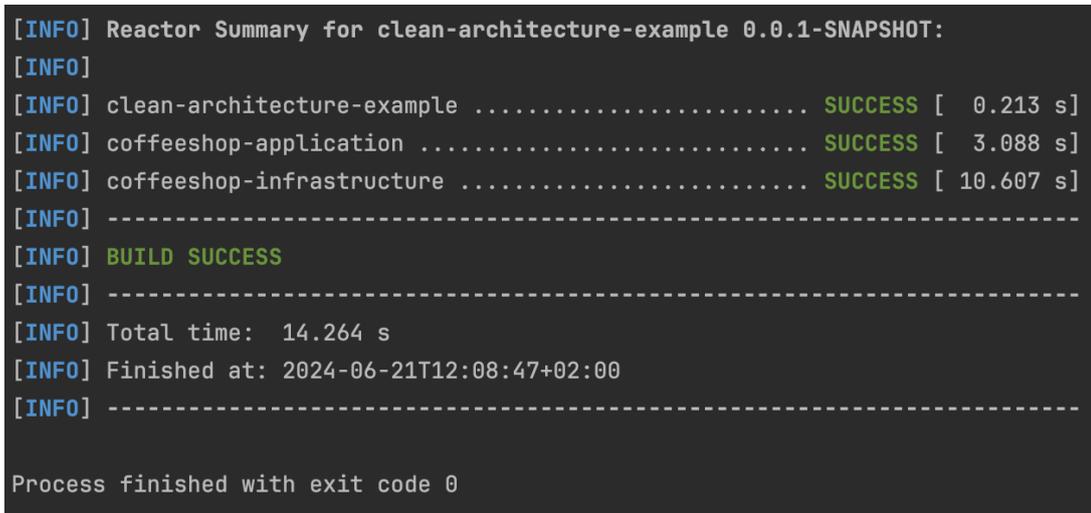
Figura 22 – Resposta retornada HTTP para criar um novo pedido



```
Body Cookies Headers (6) Test Results
Pretty Raw Preview Visualize JSON ↕
1 {
2   "id": "f7920ab7-e621-4c6f-9b63-79192a02b24d",
3   "localConsumoPedido": "LOJA",
4   "items": [
5     {
6       "bebida": "LATTE",
7       "tipoLeite": "INTEGRAL",
8       "tamanhoBebida": "GRANDE",
9       "quantity": 1
10    }
11  ],
12  "cost": 5.0
13 }
```

Fonte: Adaptado pelo autor

Figura 23 – Build do projeto



```
[INFO] Reactor Summary for clean-architecture-example 0.0.1-SNAPSHOT:
[INFO]
[INFO] clean-architecture-example ..... SUCCESS [ 0.213 s]
[INFO] coffeeshop-application ..... SUCCESS [ 3.088 s]
[INFO] coffeeshop-infrastructure ..... SUCCESS [ 10.607 s]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 14.264 s
[INFO] Finished at: 2024-06-21T12:08:47+02:00
[INFO] -----

Process finished with exit code 0
```

Fonte: Adaptado pelo autor

por se tornar plenamente funcional e operacional. Essa transição é uma demonstração vívida da versatilidade e eficácia do modelo em auxiliar no desenvolvimento de soluções práticas. O fato de ter evoluído para um estado totalmente funcional destaca não apenas a capacidade do ChatGPT em fornecer orientação útil, mas também a habilidade dos desenvolvedores em transformar conceitos iniciais em realidades tangíveis.

5 Considerações Finais

Neste trabalho, foram realizadas diversas atividades visando atingir o objetivo principal de explorar e aprimorar o processo de desenvolvimento de software no contexto empresarial. Primeiramente, foi conduzida uma extensa revisão bibliográfica para identificar as ferramentas e metodologias mais atuais e eficazes no mercado. Em seguida, foram selecionados e implementados frameworks como Spring Boot para a construção de aplicativos Java robustos e escaláveis utilizando o ChatGPT como ferramenta durante esse processo. Além disso, ferramentas de teste e documentação de APIs, como Postman, e *frameworks* de testes automatizados, como *JUnit*, foram integrados ao processo de desenvolvimento para garantir a qualidade e a eficiência do software produzido.

Posteriormente, aplicamos essas tecnologias em projetos reais, permitindo uma avaliação prática de seu impacto. A implementação prática envolveu a criação de um ambiente de desenvolvimento controlado, onde foram realizados testes de unidade, integração e aceitação. Os resultados dessas atividades evidenciaram uma melhoria na eficiência do desenvolvimento quanto na qualidade do software entregue, ressaltando que o conhecimento específico para o uso do ChatGPT como ferramenta no processo de desenvolvimento de software, confirmando as hipóteses levantadas durante a revisão bibliográfica.

A metodologia utilizada consistiu em uma abordagem prática e teórica combinada. A revisão bibliográfica forneceu uma base sólida de conhecimento, enquanto a implementação prática permitiu a validação desses conhecimentos em um ambiente real. Essa abordagem mista garantiu que as conclusões do trabalho fossem bem fundamentadas e aplicáveis no contexto empresarial.

No entanto, alguns pontos fracos foram identificados ao longo do desenvolvimento do trabalho. A limitação de tempo para a implementação prática e a restrição do estudo a um conjunto específico de ferramentas podem inviabilizar a generalização dos resultados para outros contextos ou tecnologias. Além disso, a dependência de recursos específicos de hardware e software utilizados pode limitar a replicabilidade dos resultados.

Para atividades futuras, planeja-se a expansão do estudo para incluir uma gama mais ampla de ferramentas e metodologias, como Kanban, bem como a aplicação em diferentes contextos empresariais para validar a generalização das conclusões. Além disso, será importante realizar estudos longitudinais para avaliar o impacto dessas ferramentas no longo prazo, assim como a integração de novas tecnologias emergentes no campo do

desenvolvimento de software. Essas atividades futuras contribuirão para a consolidação das práticas recomendadas e para a contínua melhoria do processo de desenvolvimento de software.

Referências

AGGARWAL, K. et al. Has the future started? the current growth of artificial intelligence, machine learning, and deep learning. *Iraqi Journal for Computer Science and Mathematics*, 2022. Disponível em: <<https://www.iasj.net/iasj/download/cebfd60eb11898a>>. Acesso em: 1 abr 2024. Citado na página 14.

BROWN, T. B. et al. Language models are few-shot learners. *arXiv:2005.14165*, 2020. Disponível em: <<https://arxiv.org/abs/2005.14165>>. Citado na página 25.

CÂNDIDO, A. K. R. Uma revisão sistemática de estudos secundários sobre práticas ágeis de desenvolvimento de software. *Pontifícia Universidade Católica De Goiás*, 2022. Citado 3 vezes nas páginas 30, 32 e 33.

CAPOZZI, B. Do invisível ao inevitável: a ascensão incessante da ia em nossas vidas. *Olhar Digital*, 2023. Disponível em: <<https://olhardigital.com.br/2023/10/25/prodo-invisivel-ao-inevitavel-a-ascensao-incessante-da-ia-em-nossas-vidas>>. Acesso em: 31 mar 2024. Citado na página 13.

CHAVES, L. C. et al. Construção de modelo para apoiar o processo de desenvolvimento de um sistema de apoio à decisão. *JISTEM-Journal of Information Systems and Technology Management*, 2020. Citado na página 37.

COUTINHO, E. F.; BEZERRA, C. I. Simulação de alocação de recursos em projetos de desenvolvimento de software utilizando teoria das filas. *Anais do III Workshop em Modelagem e Simulação de Sistemas Intensivos em Software SBC*, p. 30–39, 2021. Citado 3 vezes nas páginas 29, 30 e 31.

DEVLIN, J. et al. Bert: Pre-training of deep bidirectional transformers for language understanding. *arxiv*, 2019. Citado na página 25.

DOGAN, O. et al. A systematic review on ai/ml approaches against covid-19 outbreak. *Complex Intelligent Systems*, 2021. Disponível em: <<https://link.springer.com/content/pdf/10.1007/s40747-021-00424-8.pdf>>. Acesso em: 31 mar 2024. Citado na página 15.

GOES, B. P. et al. Estudo de caso: metodologias ágeis scrum e asd aplicadas em empresas de desenvolvimento de software e projetos. *Congresso Brasileiro de Ciências e Saberes Multidisciplinares*, p. 1–9, 2022. Citado 2 vezes nas páginas 35 e 48.

GOLD, E. *The History of Artificial Intelligence from the 1950s to Today*. 2023. Disponível em: <<https://www.freecodecamp.org/news/the-history-of-ai/#the-ai-winter-of-the-1980s>>. Acesso em: 2 abr 2024. Citado 2 vezes nas páginas 17 e 18.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep learning*. MIT Press, 2016. Disponível em: <<https://arxiv.org/abs/1907.11692>>. Citado na página 26.

GRANDO, N. *Neurônios e Redes Neurais Artificiais*. 2022. Disponível em: <<https://neigrando.com/2022/03/03/neuronios-e-redes-neurais-artificiais/>>. Acesso em: 10 abr 2024. Citado na página 20.

- HAENLEIN, M.; KAPLAN, A. A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. 2019. Disponível em: <https://www.researchgate.net/publication/334539401_A_Brief_History_of_Artificial_Intelligence_On_the_Past_Present_and_Future_of_Artificial_Intelligence>. Acesso em: 1 abr 2024. Citado 2 vezes nas páginas 17 e 18.
- KAI-FUU, L. *Inteligência Artificial*. 1. ed. [S.l.: s.n.], 2019. Citado 3 vezes nas páginas 13, 14 e 15.
- KHURANA, D. et al. Natural language processing: state of the art, current trends and challenges. *Springer Science*, 2022. Citado na página 22.
- KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 2012. Disponível em: <<https://arxiv.org/abs/1907.11692>>. Citado na página 26.
- LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Advances in neural information processing systems*, 2015. Disponível em: <<https://arxiv.org/abs/1907.11692>>. Citado na página 26.
- LIU, Y. et al. Roberta: A robustly optimized bert pretraining approach. *arXiv:1907.1169*, 2019. Disponível em: <<https://arxiv.org/abs/1907.11692>>. Citado na página 25.
- LIU, Z.; LIN, Y.; SUN, M. *Representation Learning for Natural Language Processing*. 2. ed. [S.l.: s.n.], 2023. ISBN 978-981-99-1599-6. Citado na página 15.
- MAYNARD, A. D. Navigating the fourth industrial revolution. *nature nanotechnology*. p. 1005–1006, 2015. Citado 4 vezes nas páginas 29, 30, 31 e 33.
- MEDEIROS, A. Uma adaptação de metodologia Ágil para o desenvolvimento de software educativo. *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, p. 379, 2019. Citado na página 36.
- NETO, J. G. Metodologias ágeis em uma microempresa de desenvolvimento de softwares: um estudo de caso com o uso do scrum. *Trabalho de Conclusão de Curso (Especialização em Gestão da Tecnologia da Informação e Comunicação - Universidade Tecnológica Federal do Paraná)*, 2019. Citado na página 30.
- OSTERWEIL, L. Software processes are software too. p. 323–344, 2011. Citado 4 vezes nas páginas 34, 35, 36 e 37.
- PONTES, T. B.; ARTHAUD, D. D. B. Metodologias ágeis para o desenvolvimento de softwares. *Revista Ciência e Sustentabilidade*, v. 4, 2018. Citado na página 47.
- RAFFEL, C. et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, p. 1–67, 2020. Disponível em: <<https://jmlr.org/papers/v21/20-074.html>>. Citado na página 25.
- RUSSELL, S.; NORVIG, P. *Inteligência Artificial*. 3. ed. [S.l.: s.n.], 2013. Citado 2 vezes nas páginas 14 e 16.
- SAKURAI, R.; ZUCHI, J. D. As revoluções industriais até a indústria 4.0. *Revista Interface Tecnológica*, p. 480–491, 2018. Citado 2 vezes nas páginas 29 e 30.

SANCHES, F. de A. Metodologias Ágeis no desenvolvimento de software. boletim do gerenciamento. v. 8, p. 35–44, 2019. Citado na página 47.

SCHWAB, K. *A quarta revolução industrial*. [S.l.]: Edipro, 2019. Citado 4 vezes nas páginas 29, 30, 33 e 34.

SILVA, P. R. da; SANTOS, M. R. dos; SHIBAO, F. Y. Desenvolvimento de softwares. *CMMI e Metodologias Ágeis. Revista Livre de Sustentabilidade e Empreendedorismo*, v. 4, p. 157–184, 2019. Citado na página 48.

SILVA, W. S. da. Criação de um modelo de processo híbrido aplicado no desenvolvimento de um software baseado na integração das metodologias de desenvolvimento de software. *Trabalho de Conclusão de Curso, apresentado para obtenção do grau de Bacharel no Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense*, 2020. Citado 3 vezes nas páginas 34, 36 e 37.

SMITH, C. et al. The history of artificial intelligence. 2006. Disponível em: <<http://courses.cs.washington.edu/courses/csep590/06au/projects/history-ai.pdf>>. Acesso em: 1 abr 2024. Citado na página 16.

VASWANI, A. et al. Attention is all you need. *ACM - Association for Computing Machinery*, 2017. Disponível em: <<https://dl.acm.org/doi/10.5555/3295222.3295349>>. Citado 2 vezes nas páginas 24 e 27.

WIEGERS, K.; BEATTY, J. *Software requirements*. [S.l.]: Pearson Education, 2013. Citado na página 37.



PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
GABINETE DO REITOR

Av. Universitária, 1069 • Setor Universitário
Caixa Postal 86 • CEP 74605-010
Goiânia • Goiás • Brasil
Fone: (62) 3946.1000
www.pucgoias.edu.br • reitoria@pucgoias.edu.br

RESOLUÇÃO nº 038/2020 – CEPE

ANEXO I

APÊNDICE ao TCC

Termo de autorização de publicação de produção acadêmica

O(A) estudante Victor Junio Lisboa Costa do Curso de Ciência da computação, matrícula: 20182002800527, telefone: 62 998305106 e-mail: victorjuniocosta@gmail.com, na qualidade de titular dos direitos autorais, em consonância com a Lei nº 9.610/98 (Lei dos Direitos do Autor), autoriza a Pontifícia Universidade Católica de Goiás (PUC Goiás) a disponibilizar o Trabalho de Conclusão de Curso intitulado: ChatGPT: Uma análise da ferramenta aplicada ao desenvolvimento de software, gratuitamente, sem ressarcimento dos direitos autorais, por 5 (cinco) anos, conforme permissões do documento, em meio eletrônico, na rede mundial de computadores, no formato especificado (Texto(PDF); Imagem (GIF ou JPEG); Som (WAVE, MPEG, AIFF, SND); Vídeo (MPEG, MWV, AVI, QT); outros, específicos da área; para fins de leitura e/ou impressão pela internet, a título de divulgação da produção científica gerada nos cursos de graduação da PUC Goiás.

Goiânia, 26 de Março de 2024.

Assinatura do autor: Victor Costa

Nome completo do autor: Victor Junio Lisboa Costa

Assinatura do professor-orientador: [Assinatura]

Nome completo do professor-orientador: Lucília Gomes Ribeiro